Introducere

În activitatea îndelungată pe care o au, absolvenţii specialişti în Cibernetică, Statistică şi Informatică Economică sunt nevoiţi să lucreze cu o multitudine de limbaje de programare. Adaptarea lor rapidă la diverse cerinţe de programare necesită o bună mobilitate intelectuală în domeniu. De aceea, cunoaşterea unei "metodologii" de învăţare a unui limbaj de programare este absolut necesară şi determină diminuarea stresului programatorilor şi creşterea capacităţii lor de concepţie a programelor. Stăpânirea mecanismelor, principiilor şi etapelor de învăţare a unui nou limbaj de programare este o ştiinţă.

Autorii acestei lucrări consideră că mai importante decât limbajul în sine sunt principiile generale ale programării. De aceea, cursurile universitare susţinute de ei pun accentul pe teoria programării, însoţită de rezolvarea de către studenţi a unui număr cît mai mare de aplicaţii rulate pe calculator.

Învăţarea unui limbaj de programare se poate face în două situaţii: cel care învaţă ia prima dată contact cu acest domeniu sau are deja o anumită experienţă.

Cele două situaţii se abordează didactic în mod cu totul diferit. În prima situaţie, studentul este acomodat cu logica programării; se lămuresc principiile realizării operaţiilor din algoritmi, se clarifică terminologia de specialitate, se prezintă un prim limbaj de programare, se determină plăcerea lucrului la calculator etc.

Cea de-a doua situaţie este mai comodă din punct de vedere didactic. Trebuie să se continue adâncirea logicii programării, trebuie să se lămurească principiile generale ale programării, modul fizic de execuţie a programelor, funcţiile compilării etc. Ştiinţa învăţării limbajelor de programare se bazează, în această situaţie, pe cunoaşterea principiilor generale ale programării şi a limbajului de programare deja cunoscut.

Raportarea la cunoştinţele anterioare este cea mai bună metodă de învăţare a unui nou limbaj. În această lucrare se va considera cunoscut limbajul Pascal (limbajul clasic de învăţare a programării) şi se face trecerea la limbajul C (limbaj folosit – împreună cu extensiile sale – pentru programarea în sine).

În general, în studiul unui limbaj de programare se disting două etape:

a) studiul elementelor de bază şi scrierea unor programe de iniţiere (*studiul pe orizontală*). Scopul acestei etape este de a ajunge rapid la scrierea unor programe simple cu intrări/ieşiri de la tastatură/monitor. Elementele studiate în această etapă sunt:

1	**Elemente de cunoaştere a limbajului:** cum a apărut? cine l-a conceput? ce standard s-a adoptat? etc.
2	**Construcţii de bază**: setul de caractere, identificatorii, comentariile, structura generală a programului etc.
3	**Tipuri de date:** simple, structurate, statice, dinamice etc.
4	**Expresii:** aritmetice, logice etc.
5	**Instrucţiuni pentru realizarea structurilor fundamentale**
6	**Operaţii de intrare/ieşire cu tastatura/ monitorul**

b) studiul elementelor profesionale ale limbajului (*studiul pe verticală*). Scopul acestei etape este acela de a intra în toate elementele de detaliu ale limbajului.

7	**Subprograme**	**Adâncirea studiului etapelor din etapa precedentă**
8	**Fişiere**	
9	**Structuri dinamice**	
10	**Obiecte**	
11	**Elemente de tehnica programării**	

Prezenta lucrare conţine, pe lângă partea teoretică, şi o multitudine de aplicaţii, acoperind punctele 2–9 din problematica prezentă în tabelele anterioare.

În text, simbolul ***(A)*** va marca asemănările dintre limbajele *Pascal* şi *C*, iar ***(D)*** va marca deosebirile.

Bucureşti, februarie 2003 Autorii

1. Apariția și evoluția limbajului C

Apariția limbajului *C* a fost precedată și anunțată de diverse încercări de a construi un limbaj care să corespundă cerințelor apărute o dată cu dezvoltarea calculatoarelor: portabilitate, ușurință în folosire, libertate de exprimare, text sursă compact etc. Motorul cercetărilor a fost dezvoltarea unui nou sistem de operare: UNIX. Evoluția limbajului și a sistemului de operare a avut loc în paralel, fiecare impulsionând dezvoltarea celuilalt. Se consideră că *C* a fost creat special pentru sistemele de operare din familia UNIX. De fapt *C* este un rezultat al experienței acumulate în aceste cercetări, chiar dacă este considerat un limbaj *de autor*.

În 1969 a fost lansat limbajul *BCPL* (*B*asic *C*ombined *P*rogramming *L*anguage), conceput la Universitatea Cambridge. Era un limbaj dependent de structura mașinii pe care era folosit și nu avea conceptul de *tip de dată* – datele se reprezentau binar, pe cuvinte de memorie. În multe cercuri anul 1969 este considerat anul apariției limbajului *C*.

În 1970 a apărut o formă mai avansată, limbajul *B*, utilizat sub sistemul de operare al calculatoarelor DEC PDP. Numele limbajului este o prescurtare a acronimului BCPL.

În 1978 Dennis Ritchie și Brian Kernighan și-au finalizat cercetările publicând *The C Programming Language* (Limbajul de programare C), care este limbajul cunoscut astăzi. Limbajul a fost denumit astfel pentru că *C este litera care urmează după B*. Lucrarea lui Kernighan și Ritchie este considerată momentul oficial al apariției limbajului *C*.

După apariția limbajului *C* au existat încercări de îmbunătățire, dar fără ca vreuna să producă un impact asemănător. Dennis Ritchie a propus diverse variante, mulate pe noile versiuni de UNIX (între acestea *K&R C*, creat pentru sistemul UNIX de pe mașinile DEC PDP).

În perioada următoare, pornind de la specificațiile limbajului, s-au creat numeroase compilatoare *C*, fiecare aducând propriile diferențe față de limbajul inițial, ajungându-se la variante foarte mult diferite unele față de altele. În acest fel s-a pus în pericol ideea inițială de *portabilitate*. Ca urmare, în 1988 Comitetul pentru limbajul C al Institutului Național de Standardizare al SUA (ANSI) a impus

un standard cunoscut drept *ANSI C*. Acesta este implementat în toate versiunile de compilatoare ulterioare. O dată cu acest standard legătura implicită cu sistemul UNIX a dispărut, limbajul fiind disponibil pe orice platformă.

În paralel cu activitatea comitetului ANSI C, au existat şi alte dezvoltări. Bjarne Stroustrup de la AT&T a propus extensia *C++* (*C* îmbunătățit, extinzând semnificația operatorului de incrementare ++ din *C*). Impactul extensiei este aproape la fel de mare ca al limbajului inițial, astfel încât ea a devenit cea mai populară formă a limbajului, fiind prezentă în toate implementările curente.

Pornind de la *C++* s-au încercat şi alte extensii. Joel E. Richardson, Michael J. Carrey şi Daniel T. Schuh de la Universitatea Wisconsin Madison au propus limbajul *E*. Acesta este proiectat pentru crearea de programe de sistem, în general, şi de sisteme de gestiune a bazelor de date, în particular. Una din facilități este crearea şi utilizarea *obiectelor persistente*.

Bell Laboratories a propus în paralel limbajul *O*, echivalent limbajului *E*.

Pentru a veni în întâmpinarea soluțiilor de calcul distribuit, a fost creat limbajul *Avalon/C++*.

Aventura dezvoltării limbajelor de programare continuă.

2. Construcții de bază ale limbajului

i. Identificatori. Identificatorii sînt succesiuni de litere (mici sau mari, din alfabetul englez), cifre şi _ (liniuța de subliniere – caracterul *underscore*). Primul caracter trebuie să fie literă sau liniuță de subliniere ***(A)*** (nu se recomandă ca primul caracter să fie _, pentru a nu se face confuzii nedorite cu identificatorii rezervați folosiți de diferite compilatoare). Lungimea unui identificator variază de la un singur caracter până la oricâte, dar se iau în considerare numai primele 32. Dacă doi identificatori au primele 32 de caractere identice, atunci ei vor fi consideraţi identici de compilator ***(A)***. Unele compilatoare permit modificarea acestei limite de lungime. Literele mari nu sînt identice cu cele mici ***(D)***.

Exemple: `a`, `b`, `c`, `abc`, `Abc`, `aBc`, `x1`, `_`, `_a` etc.
`CodProdus` nu are aceeaşi semnificaţie cu `codprodus`.

ii. Cuvinte rezervate. Există anumiți identificatori cu utilizare fixă, predefinită ***(A)***. Exemple: `auto`, `do`, `if`, `else`, `char`, `long`, `float`, `for`, `return` etc. (din *K&R C*); `const`, `enum`, `signed`, `void` etc. (din *ANSI C*); `_AL`, `_AX`, `_DX`, `_BX`, `_FLAGS` (din *Turbo C*). Aceşti identificatori sînt numiți *cuvinte rezervate.*

iii. Comentarii. Comentariile sînt secvenţe de text cu format liber (în general explicații în limbaj normal), care nu se compilează. În C acestea sînt delimitate de perechile de caractere `/*` şi `*/`.

Exemplu: `/* Acesta este un comentariu */`

Orice caracter aflat între delimitatori este ignorat de compilator. Nu se admit comentarii imbricate. La compilatoarele C++ a fost adăugată posibilitatea de a scrie comentarii pe un singur rînd. Începutul unui astfel de comentariu este marcat de perechea `//` iar sfîrşitul său este marcat de sfîrşitul rîndului (nu există marcator special). Orice caracter aflat după `//` este ignorat, pînă la trecerea pe un rînd nou.

Exemplu: `//Acesta este un comentariu`

iv. Instrucţiuni. Construcţia instrucţiunilor necesită folosirea terminatorului de instrucţiune `;` (în Pascal era separator de instrucţiuni) ***(D)***. Instrucţiunile sînt simple, structurate şi compuse. O instrucţiune compusă e formată dintr-o secvenţă de instrucţiuni şi declaraţii ***(D)*** delimitate de acolade (`{, }`). Acoladele au acelaşi rol ca BEGIN şi END din Pascal.

Formatul de scriere este liber ***(A)***. O instrucţiune poate fi scrisă pe mai multe rînduri şi pe un rînd pot fi mai multe instrucţiuni. Unele compilatoare impun o lungime maximă a rîndului de 128 de caractere.

v. Structura programului. Limbajul C este orientat pe funcţii. Ca urmare a acestei orientări, programul principal este el însuşi o funcţie (care poate avea parametri şi poate întoarce un rezultat de tip întreg): `main` (cuvînt rezervat) ***(D)***. O funcţie este compusă din antet şi corp ***(A)***.

La modul general, un program este o înşiruire de funcţii şi declaraţii, între care trebuie să existe o funcţie numită `main`. Aceasta este pusă în execuţie la lansarea programului.

Forma generală a unei funcţii este:

```
antet
corp
```

Antetul este compus din: tipul rezultatului, numele funcţiei, lista parametrilor formali: `tip nume_functie(lista parametri).`

Corpul funcţiei este e instrucţiune compusă. Structura unei funcţii va fi detaliată în alt capitol.

Pentru funcţia *main* sînt permise mai multe forme ale antetului:

```
main()
int main()
void main()
main(void)
void main(void)
int main(void)
```

În orice program există o parte de declaraţii şi una executabilă (instrucţiuni). Unele limbaje impun o separare foarte clară a acestora (Pascal, Cobol), în timp ce altele (C/C++) sînt mult mai flexibile, păstrînd doar regula că orice entitate trebuie să fie definită înainte de a fi referită. În C nu există nici un fel de delimitare a celor două părţi logice; declaraţiile pot să apară între instrucţiuni. Pentru a evita confuziile se recomandă însă ca declaraţiile să fie reunite la începutul blocului de program pentru care sînt vizibile.

Modalităţile de declarare a variabilelor, tipurilor noi de date etc. vor fi detaliate în continuare.

vi. Directive şi preprocesare. Înaintea compilării, în C se desfăşoară etapa de *preprocesare* ***(D)***. În cadrul ei se efectuează substituiri asupra textului sursă scris de programator. Prin preprocesare se asigură: inserarea de fişiere în textul

sursă, definiții şi apeluri de macro-uri, compilare condiționată. Preprocesarea este controlată prin *directive*. Acestea sînt formate din caracterul `#` urmat de un cuvînt rezervat şi eventuali parametri.

Exemplu: `#include<stdio.h>`

Cele mai folosite directive sînt `#include` şi `#define`.

• Directiva `#include` este folosită pentru includerea de fişiere cu text sursă în program. Forma ei este:

```
#include<specificator_de_fisier>
```

sau

```
#include"specificator_de_fisier"
```

Prima formă caută fişierul specificat între fişierele standard ale limbajului, folosindu-se calea descrisă în mediul de programare. A doua formă caută fişierul specificat în calea curentă (de obicei este folosită pentru a include fişiere scrise de utilizator). Dacă fişierul căutat nu este găsit se produce o eroare de compilare. Dacă fişierul este găsit, conținutul lui este inserat în program, în locul directivei care l-a invocat (aceasta este ştearsă – la limită poate fi considerată o substituire de text).

Pentru ca textul inclus să fie vizibil din tot programul, se recomandă ca directivele `#include` să fie scrise la începutul programului. Un text inclus poate să conțină la rîndul lui directive de includere care determină noi includeri de text.

Includerea de text sursă are un rol asemănător cu utilizarea unităților din Pascal, dar nu este identică, avînd o utilizare mult mai largă. Textul inclus va fi compilat ca parte componentă a programului, spre deosebire de unitățile Pascal care sînt deja compilate. Nu se poate spune că există echivalență între cele două tehnici, dar rolul lor final este acelaşi.

Exemplu: pentru a putea utiliza funcțiile de bibliotecă de intrare/ieşire trebuie inclus fişierul `stdio.h`.

```
#include<stdio.h>
```

• Directiva `#define` este folosită pentru a substitui secvențe de caractere. Forma ei generală este:

```
#define sir1 sir2
```

unde atît *sir1*, cît şi *sir2* sînt şiruri de caractere. La preprocesare, se şterge din program această directivă şi, în tot programul, se înlocuieşte secvența de caractere *sir1* cu secvența de caractere *sir2*. Secvența *sir1* este numită *nume*, iar *sir2* este numită *descriere*. Nu se face înlocuirea dacă *sir1* apare în interiorul unui literal şir de caractere sau în interiorul unui comentariu. *sir2* poate să fie descris pe mai multe rînduri, dacă la sfîrşitul fiecărui rînd (în afară de ultimul) se scrie caracterul `\` (backslash). Rolul acestuia va fi discutat în alt capitol. *sir2* poate să conțină şiruri de caractere pentru care au fost definite descrieri anterior. Acestea vor fi substituite conform definițiilor lor.

Substituirea poate fi dezactivată din punctul în care apare directiva `#undef` pînă la sfîrşitul programului sau pînă la redefinirea lui *sir1*. Directiva are forma generală:

```
#undef sir1
```

Între cele mai frecvente utilizări ale directivei `#define` se află: definirea de constante simbolice şi definirea de *macro*-uri (*macrodefiniții*) – pentru simplificarea scrierii.

Exemple:
```
#define N 10
#define M 10
#define MAX  (M+N)
#define DIM(a,b) (a)*(b)
char v[N],v1[10+DIM(5+M,6)];
char v1[10*MAX];
char m[M][N];
```

După preprocesare secvența de cod va deveni:

```
char v[10],v1[10+(5+10)*(6)];
char v1[10*(10+10)];
char m[10][10];
```

Se observă că la preprocesare nu se efectuează nici un calcul, doar substituiri de text. Preprocesorul nu „înțelege” textul pe care îl manipulează.

În exemplele de mai sus, *M* şi *N* sînt *constante simbolice* (concept prezent şi în Pascal ***(A)***), în timp ce *MAX* şi *DIM* sînt *macrodefiniții* (concept inexistent în Pascal ***(D)***).

Macrodefiniția este un nume simbolic asociat unei secvențe fixe sau variabile (cu o parte fixă şi una variabilă) de text sursă. Secvenței i se poate asocia şi o listă de parametri, caz în care are o parte variabilă. Macrodefinițiile sînt folosite pentru a uşura scrierea textului sursă. Ele sînt nume acordate unor secvențe care se repetă frecvent, identic sau cu mici variații.

Din punct de vedere al textului sursă, un macro se foloseşte la fel ca orice funcție: se apelează prin numele simbolic urmat de lista parametrilor reali, pusă între paranteze. Din punct de vedere al compilării, există o diferență fundamentală: macro-urile sînt tratate la *precompilare*. Precompilatorul şterge din textul sursă apelul macro-ului şi îl înlocuieşte chiar cu secvența respectivă. Parametrii formali ai macro-ului sînt înlocuiți cu parametrii reali, prin substituire de text (corespondența se face conform regulilor de punere în corespondență a parametrilor reali cu cei formali: unu-la-unu, de la stînga la dreapta). Se observă folosirea parantezelor în definirea macrodefiniției pentru a evita problemele legate de ordinea de precedență a operatorilor, care pot să apară la substituire. În general se recomandă ca entitățile care participă la substituire să fie cuprinse între paranteze.

Operația de substituire a numelui macrodefiniției cu descrierea sa se numeşte *expandarea macrodefiniției*.

3. Tipuri de date

Mulțimea tipurilor de date predefinite constituie o caracteristică importantă a oricărui limbaj şi un argument în alegerea unui limbaj sau altul pentru rezolvarea unei probleme.

În general, tipurile de date se clasifică în *statice* şi *dinamice*, împărțite la rîndul lor în simple şi structurate. Pornind de la clasificarea teoretică a tipurilor de date putem face trecerea de la Pascal la C:

Tabelul 3.1 Clasificarea tipurilor de date

			Pascal	**C**
Statice	simple	întregi (diverse)	Da	Da
		reale (diverse)	Da	Da
		caracter	Da	Da
		enumerativ	Da	Da
		logic	Da	Nu
	structurate	masiv	Da	Da
		şir de caractere	Da	Nu*
		mulțime	Da	Nu
		articol	Da	Da
		fişier	Da	Da
Dinamice	simple		Da	Da
	structurate		Nu	Nu

*Tipul *şir de caractere* nu există ca atare în C, dar există funcții de bibliotecă pentru tratarea şirurilor de caractere, folosind reprezentarea lor în vectori de caractere.

Ca şi în Pascal, orice dată trebuie declarată înainte de a fi folosită ***(A)***.

Din punct de vedere al modului de implementare într-un limbaj, tipurile de date se clasifică în:

- *Native* – implementate în concordanță cu nivelul fizic al procesorului. Acestea sunt: datele întregi (virgulă fixă), datele reale (virgulă mobilă). Tipurile de date native sînt prezente în toate limbajele de programare.

- *Reprezentate prin convenţie* – logice, şir de caractere etc. Limbajul C este orientat spre tipurile native, folosind puţine convenţii. Datele reprezentate prin convenţie în C au doar interpretare numerică.

Exemplu:

- Tipul *logic* nu există dar, prin convenţie, valoarea 0 (de orice tip: întreg, real, pointer) este asimilată cu *fals*, iar orice altă valoare este asimilată cu *adevărat*.
- Tipul *şir de caractere* nu există, dar, prin convenţie, se foloseşte reprezentarea în vectori de caractere.

3.1 Tipuri simple de date

În limbajul C există o serie de tipuri simple de date predefinite (tabelul 3.2), a căror lungime poate să difere de la un calculator la altul şi de la o implementare la alta. Standardul C este mai elastic decît altele. De exemplu, pentru tipul *char* este definită lungimea 1 sau cel mult lungimea tipului *int*.

Tabelul 3.2 Tipuri simple în C

Grupa de dată	Tipul	Lungime (octeţi)	Domeniu de valori	Mod de reprezentare
Întreg	[signed] char	1	-128..127 ($-2^7..2^7-1$)	Codul ASCII al caracteru-lui. Poate fi prelucrat ca un caracter sau ca întreg cu/fără semn.
	unsigned char	1	0..255 ($0..2^8-1$)	
	unsigned [int]	2	0..65535	Întreg fără semn
	[short] [int]	2	-32768..32767	Complement faţă de 2
	unsigned long	4	$0..2^{32}-1$	Întreg fără semn
	long [int]	4	$-2^{31}..2^{31}-1$	Complement faţă de 2
Real	float	4	$3.4*10^{-38}..3.4*10^{38}$	Virgulă mobilă simplă precizie
	double	8	$1.7*10^{-308}.. 1.7*10^{308}$	Virgulă mobilă dublă precizie
	long double	10	$3.4*10^{-4932}.. 3.4*10^{4932}$	Virgulă mobilă dublă precizie

Observaţie: Pentru fiecare tip predefinit din limbajul C există cuvinte rezervate, care reprezintă modificatori de tip: *unsigned*, *signed*, *long* şi *short* pentru tipul *int*; *signed* şi *unsigned* pentru tipul *char*; *long* pentru tipul *double*. Cuvintele rezervate dintre parantezele pătrate sînt opţionale şi diferitele combinaţii definesc acelaşi tip de dată.

Declararea variabilelor

În C nu există declarații implicite ***(A)***. Declararea variabilelor se face asemănător cu limbajul Pascal, prin listă de identificatori pentru care se specifică tipul. Deosebirea constă doar în sintaxă: în C se scrie întîi tipul, apoi lista de variabile ***(D)***. În plus, în C nu există secțiune specială pentru declararea variabilelor ***(D)***. Declararea se poate face oriunde, iar domeniul de valabilitate este limitat la blocul în care s-a făcut declarația.

- în Pascal: `VAR lista_variabile:tip;`
- în C: `tip lista_variabile;`

Exemple:

```
unsigned x,y,z;
float a,b,c;
char k;
```

Definirea de noi tipuri de date

Ca şi în Pascal, în limbajul C utilizatorul poate defini noi tipuri de date sau poate atribui un alt nume unui tip predefinit sau definit anterior ***(A)***. În acest scop se foloseşte cuvîntul rezervat *typedef*. Definirea se face asemănător cu limbajul Pascal, inversînd sintaxa.

- în Pascal: `TYPE nume_utilizator=descriere_tip;`
- în C: `typedef descriere_tip nume_utilizator;`

Observație: Pentru lizibilitatea textului sursă, se obişnuieşte ca numele atribuit unui tip de date să se scrie cu litere mari.

Exemple:

```
typedef int INTREG;
typedef float REAL;
```

3.2 Particularități ale unor tipuri simple de date

Tipul caracter

Tipul caracter memorează caractere ale codului ASCII, reprezentate pe un octet. Variabilele de tip caracter pot fi utilizate şi ca valori numerice (modul de utilizare se alege automat, în funcție de expresia din care face parte operandul respectiv).

Valoarea numerică folosită depinde de modul de declarare a caracterului: cu sau fără semn. Pentru caracterele cu coduri mai mici decât 128 nu se sesizează nici o diferență (se obține aceeaşi valoare şi pentru interpretarea ca virgulă fixă aritmetică şi pentru interpretarea ca virgulă fixă algebrică). Pentru caracterele cu coduri mai mari de 127, valoarea obținută este diferită.

Exemple:

- declararea şi utilizarea variabilelor de tip caracter

```
unsigned char a,b;
……
a=100;/* corect */
b='Q';/* corect */
b=81; /* corect, echivalent cu precedentul */
a=a+b;/* corect, prin context, se lucreaza numeric, cu
valoarea 81 pentru variabila b*/
```

- programul următor declară două variabile de tip caracter: *a* (cu semn) şi *b* (fără semn). Ambele variabile sînt iniţializate cu valoarea 200 şi apoi afişate atît ca întreg, cît şi ca tip caracter.

```
#include<stdio.h>
void main()
{ char a;
  unsigned char b;
  a=200; b=200;
  printf("\n\t intreg: %d \t caracter: %c",a,a);
  printf("\n\t intreg: %d \t caracter: %c",b,b);}
```

Rezultatul obţinut este următorul:

```
intreg: -56     caracter: È
intreg: 200     caracter: È
```

Se observă că, deşi au fost iniţializate şi tratate în mod identic, valoarea numerică diferă în funcţie de modul de declarare (*a* este cu semn, *b* este fără semn). Funcţia *printf* va fi prezentată ulterior.

3.3 Constante

În studiul constantelor se poate porni de la clasificarea lor teoretică.

Constante	Literali (se autoidentifică prin valoare)	Numerici	Întregi	Da
			Reali	Da
		Nenumerici	Caracter	Da
			Şir de caractere	Da
			Logic	Nu
	Constante simbolice (identificatori asociaţi constantelor)			Da

Observaţie: Ultima coloană precizează facilităţile în C.

Literalii întregi pot fi exprimați în bazele 10, 8 (folosind prefixul *0* – zero) sau 16 (folosind prefixul *0x* sau *0X*). În funcție de mărimea lor, se asociază implicit un tip întreg (și implicit un mod de reprezentare). Se încearcă întotdeauna întîi reprezentarea pe 16 biți, conform tipului *int*; dacă nu este posibilă, atunci se foloseşte reprezentarea pe 32 de biți, conform tipului *long*.

Dacă dorim să forțăm reprezentarea pe 32 de biți (pentru o valoare din domeniul tipului *int*) se adaugă sufixul *l* sau *L*. Pentru a forța tipul *fără semn* (*unsigned int* sau *unsigned long*) se foloseşte sufixul *u* sau *U*. Cele două sufixe se pot folosi împreună, în orice ordine.

Exemple: exprimarea domeniului pentru tipul *unsigned int* în cele 3 baze:

Zecimal	Octal	Hexazecimal
0÷32767	00÷077777	0x0000÷0x7fff

12345 - întreg zecimal reprezentat pe 2 octeți
-12345 - întreg zecimal reprezentat pe 2 octeți
12345L - întreg zecimal reprezentat pe 4 octeți
012345 - întreg octal reprezentat pe 2 octeți
0x1234 - întreg hexazecimal reprezentat pe 2 octeți

Exprimare externă	**Reprezentare internă**
12345	0011000000111001
123456789	00000111010110111100l101000010101
12345L	00000000000000000011000000111001

Literalii reali se pot exprima sub formă matematică (**±întreg.fracție**) sau ştiințifică (**±întreg.fracțieE±exponent**). Semnul + poate lipsi (este implicit), iar **e** este echivalent cu **E**. Din exprimare poate să lipsească fie partea fracționară, fie partea întreagă, fie partea întreagă şi exponentul (inclusiv litera *e*).

Exemple:

Exprimare externă	**Valoare**
1.56	1,56
177e-1	17,7
15.5E3	15500 ($15,5x10^3$)
453.	453,0
.34	0,34
.1E-3	0,0001 ($0,1x10^{-3}$)
123.456e-4	0,123456 ($123,456x10^{-4}$)

Literalii reali se reprezintă intern în virgulă mobilă dublă precizie, pe 64 de biți (tipul *double*). Pentru a forța reprezentarea în simple precizie (tipul *float*) se adaugă sufixul *f* sau *F*. Pentru a forța reprezentarea pe 80 de biți (tipul *long double*) se foloseşte sufixul *l* sau *L*. Cele două sufixe nu se pot folosi împreună.

Literalii de tip caracter se reprezintă intern prin codul ASCII al caracterului respectiv, pe un octet. Exprimarea externă depinde de caracterul respectiv. Literalii de tip caracter pot participa în expresii cu valoarea lor numerică, aşa cum s-a arătat anterior.

Exprimarea externă a unui caracter imprimabil se face prin caracterul respectiv inclus între apostrofuri. Excepţie fac caracterele cu semnificaţie specială în C: *apostrof*, *ghilimele* şi *backslash*. Pentru acestea, între apostrofuri se include un caracter *backslash*.

Exemple: `'B'`, `'b'`, `'7'`, `' '` (spaţiu), `'*'`, `'\\'` (caracterul backslash, cod ASCII 92), `'\''` (caracterul apostrof, cod ASCII 39), `'\"'` (caracterul ghilimele, cod ASCII 34).

Reprezentarea folosind caracterul *backslash* este numită *secvenţă escape*. Secvenţele escape sînt folosite pentru a reprezenta caracterele de control (coduri ASCII între 0 şi 31).

Exemple:

Literal	Cod ASCII	Denumire	Utilizare
`'\a'`	7	BEL	Emite un sunet
`'\b'`	8	BS	Revenire cu un spaţiu (backspace)
`'\t'`	9	HT	Tab orizontal (9) spaţii
`'\n'`	10	LF	*Newline*, corespunde perechii CR/LF – rînd nou
`'\v'`	11	VT	Tab vertical
`'\f'`	12	FF	Pagină nouă la imprimantă (*form feed*)
`'\r'`	13	CR	Poziţionare la începutul rîndului (*carriage return*)

Pentru a reprezenta caracterele codului ASCII extins (coduri 128-255) se pot folosi numai secvenţele escape construite astfel:

`'\ddd'`, unde `d` este o cifră din sistemul de numeraţie octal (0÷7). Construcţia `ddd` este considerată implicit ca fiind codul ASCII al caracterului, reprezentat în baza 8. Nu este nevoie ca ea să fie precedată de un zero nesemnificativ. Această construcţie poate fi folosită pentru a reprezenta orice caracter al setului ASCII.

Exemple: `'\a'` şi `'\7'` reprezintă caracterul BEL, `'\b'` şi `'\10'` reprezintă caracterul BS, `'\"'` şi `'\42'` reprezintă caracterul ghilimele, `'\377'` reprezintă caracterul cu codul ASCII 255.

La iniţializarea unei variabile de tip caracter se poate folosi oricare din variantele de reprezentare descrise anterior sau orice valoare numerică (întreagă sau reală). În acest ultim caz, din reprezentarea internă a literalului numeric se iau în considerare primii 8 biţi care sînt interpretaţi ca un cod ASCII, obţinîndu-se valoarea care se atribuie.

Literali de tip şir de caractere

Un literal de tip şir de caractere este un şir de zero sau mai multe caractere, delimitate prin ghilimele (ghilimelele nu fac parte din şirul respectiv). În interiorul şirului se pot folosi secvenţe escape pentru a reprezenta diferite caractere. Un literal de tip şir de caractere poate fi scris pe mai multe rînduri. Pentru a semnala că un literal continuă pe rîndul următor se scrie caracterul `\` la sfîrşitul rîndului curent.

Un şir de caractere este reprezentat intern prin codurile ASCII ale caracterelor, cîte unul pe fiecare octet, la sfîrşit adăugîndu-se caracterul *nul* (cod ASCII 0 – `'\0'`). Caracterul *nul* nu poate să facă parte dintr-un şir, el avînd rolul de terminator de şir. În reprezentarea internă, un şir de caractere ocupă cu un octet mai mult decît numărul de caractere din componenţa sa.

Exemplu:

```
"Limbajul C este destul de usor, \
daca \"stii\" limbajul Pascal"
```

reprezintă şirul *Limbajul C este destul de usor, daca "stii" limbajul Pascal*.

Observaţii

Folosirea secvenţelor escape poate duce la efecte nedorite. Dacă după caracterul `\` urmează o cifră, se vor lua în considerare şi următoarele cifre (maxim 3) pentru a compune un cod ASCII în baza 8. Literalul `"m\9p"` reprezintă un şir format din: caracterul *m*, caracterul *tab orizontal* şi caracterul *p*. Literalul `"m\90"` reprezintă şirul *mH*, adică este format din caracterul *m* şi caracterul cu codul ASCII 72 (090 în baza 8). Literalul `"m\1751"` reprezintă şirul *m}1* (175 în baza 8 este codul ASCII al caracterului *}*). Dacă se doreşte reprezentarea şirului format din: caracterul *m*, caracterul *tab orizontal* şi caracterul *0* (de exemplu), trebuie să scriem `"m\9\60"` (unde \60 este reprezentarea caracterului *0*).

Constante simbolice (cu nume)

Constantele simbolice din C se definesc folosind directiva `#define`, aşa cum s-a arătat anterior.

C	Pascal
`#define nume_const valoare`	`CONST nume_const=valoare;`

Exemple:

```
#define pi 3.14159
pi=7.9; /*eroare la compilare*/
```

Variabile iniţializate la compilare (constantele cu tip)

C	Pascal
`tip nume_const=valoare;`	`CONST nume_const:tip=valoare;`

Observaţie: În ambele limbaje, constantele cu tip joacă rol de variabile care se iniţializează cu o valoare în faza de compilare, ele putînd să-şi modifice valoarea pe parcursul execuţiei programului ***(A)***. Acest comportament este indicat mai bine de modul de definire din C.

Exemple:

```
float pi=3.14159;
pi=7.9; /*nu este eroare*/
```

Constantele "obiect" există doar în limbajul C. Ele sînt variabile iniţializate la declarare, pentru care se rezervă memorie, dar conţinutul lor nu poate fi modicat pe parcursul programului. Ele se declară folosind modificatorul *const*:

```
const tip nume_const=valoare;
```

Exemplu:

```
const float PI=3.14159;
```

dacă se încearcă o atribuire (de exemplu `PI=7`), se generează eroare.

3.4 Tipuri structurate

Tipul masiv

Masivul este o structură de date omogenă, cu acces direct la elementele sale. În limbajul C un masiv se declară folosind o construcţie de forma:

```
tip nume[d1][d2]…[dn];
```

unde *tip* este tipul comun elementelor masivului iar *d1*, *d2*, *dn* sînt expresii constante (care pot fi evaluate la compilare) care indică numărul de elemente de pe fiecare dimensiune. Se acceptă oricîte dimensiuni, cu condiţia ca structura în ansamblul ei să nu depăşească 64Ko.

Exemple:

```
int b[10];       /* vector cu 10 componente intregi*/
float a[10][20]; /* matrice cu 10 linii si 20 coloane */
```

O structură de tip masiv este memorată, ca şi în Pascal, lexicografic (pe linii). La declararea masivelor se poate face şi iniţializarea lexicografică a acestora. Numărul valorilor care se iniţializează trebuie să fie mai mic sau egal cu numărul maxim al elementelor masivului. În cazul în care se face iniţializarea integrală a masivului, dimensiunile a căror valoare nu intră în generarea funcţiei rang pot lipsi.

Exemple:

```
int b[10]={2,3,0,5,6,7,3,6,8,5};
        /*vectorul contine valorile 2 3 0 5 6 7 3 6 8 5*/
int b[10]={2,3,0,5,6,7,3,6,8,5,6};
       /*se va semnala eroare la compilare, deoarece sînt mai multe
       valori pentru initializare*/
int b[10]={2,3,0,5,6,7};
       /* vectorul contine valorile 2 3 0 5 6 7 0 0 0 0*/
```

```
int b[]={1,2,3,4,5,6,7,8,9,0};
       /*se rezerva spatiu pentru 10 elemente, care sint
initializate
cu valorile 1,2,3,4,5,6,7,8,9,0*/
float a[5][3]={{1,2,3},{1,2,3},{1,2,3}};
       /* matricea contine valorile 1 2 3 1 2 3 1 2 3 0 0 0 0 0 0*/
float a[5][3]={{1,2},{1,2,3},{1}};
        /* matricea contine valorile 1 2 0 1 2 3 1 0 0 0 0 0 0 0 0*/
```

Observație: În limbajul C există trei tipuri de variabile: *globale*, *statice* şi *automatice*. Variabilele globale şi statice neinițializate au implicit valoarea inițială egală cu zero. Variabilele automatice neinițializate au valoarea inițială imprevizibilă. În Pascal, orice variabilă neinițializată are, implicit, o valoare imprevizibilă.

Masivul poate fi referit **global** prin numele său, care reprezintă în Pascal şi C adresa simbolică de început a zonei de memorie în care sînt memorate elementele acestuia. Elementele se referă **direct**, prin numele masivului şi indicele (indicii) corespunzători. De exemplu:

`a[5][1]` - elementul de pe linia a 6-a, coloana a 2-a;

`b[5]` - al 6-lea element din vector.

În C, primul indice de pe fiecare dimensiune are valoarea 0, iar compilatorul nu verifică corectitudinea indicilor (în sensul de depăşire a dimensiunilor masivului la referirea elementelor). Regăsirea elementelor se face prin calcularea adreselor lor relativ la adresa de început a masivului. Aceste calcule vor fi detaliate în capitolul despre pointeri.

Exemple:

```
float x[100];
x[0]=5.987;     /*primul element*/
x[99]=7.4539;   /*ultimul element*/
```

Pentru expresiile `x[110]` şi `x[-20]` compilatorul nu semnalează eroare, dar zona conține o valoare imprevizibilă. Mai mult, zona este, probabil, atribuită unei alte variabile, iar înscrierea de valori în aceste zone va corupe valoarea acelor variabile.

Tipul şir de caractere

În limbajul C nu există tipul şir de caractere predefinit (cum este în Pascal tipul *string*), dar şirurile de caractere pot fi prelucrate, reprezentarea lor fiind convențională (masive unidimensionale cu elemente de tipul *char*). Şirurile de caractere se reprezintă intern printr-o succesiune de octeți în care sînt memorate codurile ASCII ale caracterelor şirului. Ultimul octet conține caracterul NULL (cod ASCII 0 – sau ASCIIZ) şi marchează sfîrşitul şirului. Pentru memorarea unui şir de *n* caractere sînt necesare *n+1* octeți. Marcatorul de sfîrşit nu face parte din şir şi este tratat ca atare de funcțiile care lucrează cu şiruri de caractere.

Exemple:

1. `char s[10]="Limbajul C";`

0	1	2	3	4	5	6	7	8	9	10
L	i	m	b	a	j	u	l		C	0x00

2. `char c[15]= "Limbajul C";`

L	i	m	b	a	j	u	l		C	0x00	Spaţiu nefolosit			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

3. `char[]="este";`
4. declaraţia `char x[4]="ABC";`
   este echivalentă cu `char x[4]={"A","B","C",0x00};`

Exemple de operaţii eronate:

1. `char c[3]='Limbajul C';` - nu este constantă de tip şir
2. `char c[14]; c="Limbajul C";` - în C nu este definită operaţia de atribuire

Pentru prelucrarea şirurilor de caractere limbajul C pune la dispoziţie funcţii definite în biblioteca standard *string.h*. Cele mai uzuale sînt prezentate în tabelul următor:

Apelul funcţiei	**Acţiunea**
`strcmp(sir1,sir2);`	compară *sir1* cu *sir2* şi returnează rezultat întreg: negativ dacă *sir1<sir2*; 0 dacă *sir1=sir2*; pozitiv dacă *sir1>sir2*
`stricmp(sir1,sir2);`	idem, ignorînd diferenţele dintre literele mici şi literele mari (i – ignore)
`strncmp(sir1,sir2,n);`	idem, dar comparaţia se face pentru cel mult primii *n* octeţi
`strnicmp(sir1,sir2,n);`	idem, ignorînd diferenţele dintre literele mici şi literele mari, comparaţia făcîndu-se pentru cel mult *n* octeţi
`strcat(dest,sursa);`	concatenează şirul *sursa* la sfîrşitul şirului *dest*
`strncat(dest,sursa,n);`	concatenează primii *n* octeţi din şirul *sursa* la sfîrşitul şirului *dest*
`strcpy(dest,sursa);`	copiază şirul *sursa* în şirul *dest*
`strncpy(dest,sursa,n);`	copiază primii *n* octeţi din şirul *sursa* în şirul *dest*
`strlen(sir);`	returnează lungimea şirului *sir*

Observaţie: *sir1*, *sir2*, *sir*, *sursa* şi *dest* sînt adresele simbolice de început ale şirurilor.

Funcţii care lucrează cu şiruri de caractere mai sînt definite în biblioteca standard *stdlib.h*: *atof* (necesită şi includerea bibliotecii standard *math.h*), *atoi*, *atol*, *itoa*, *ltoa*. Acestea sînt funcţii de conversie.

De asemenea, în biblioteca *stdio.h* există definite funcții de intrare/ieşire pentru şiruri de caractere.

Tipul articol

Articolul este o structură de date eterogenă, cu acces direct la elementele sale. Tipul articol din C (denumit în continuare *structură*, conform terminologiei C) este asemănător tipului *record* din Pascal şi are următoarea sintaxă:

```
struct nume_tip_art {lista_cimpuri}var1,var2,…,varn;
```

unde *nume_tip_art*, *var1*, *var2*,…, *varn* sînt nume care pot lipsi, dar nu toate deodată. Dacă *nume_tip_art* lipseşte, atunci măcar *var1* trebuie să fie prezent. Dacă *var1*, *var2*,…, *varn* lipsesc, atunci *nume_tip_art* trebuie să fie prezent. În continuare, *nume_tip_art* este un tip nou de date, iar *var1*, *var2*,…, *varn* sînt variabile de tipul *nume_tip_art*.

Oricare din variabilele de tip structură poate să fie un masiv cu elemente de tip structură, dacă se modifică astfel: în loc de *var1* (de exemplu) se scrie *var1[dim1][dim2]…[dimn]*.

Lista_cîmpuri este o înşiruire de declarații de cîmpuri, asemănătoare declarațiilor de variabile (de forma *tip nume_cîmp*). Cîmpurile unei structuri pot fi variabile simple, masive sau chiar alte structuri.

O variabilă de tip *nume* poate fi declarată şi ulterior definirii structurii:

```
struct nume var1;
```

Descrierea anterioară constituie o definire implicită de nou tip de dată. Este posibilă definirea explicită de nou tip de dată, adăugînd cuvîntul rezervat *typedef* în fața declarării.

Exemplu: definirea tipului de dată *număr complex*, a unei variabile de acest tip şi a unui masiv unidimensional cu elemente de tip *complex* se poate face în oricare din următoarele variante (pentru un număr complex se va reține partea reală şi partea imaginară):

a)
```
struct COMPLEX{float r,i;}a,b[100];
```

b)
```
struct COMPLEX{float r,i;};
struct COMPLEX a,b[100];
```

c)
```
struct COMPLEX{float r,i;};
COMPLEX a,b[100];
```

d)
```
struct {float r,i;}COMPLEX;
COMPLEX a,b[100];
```

e)
```
typedef struct {float r,i;} COMPLEX;
COMPLEX a,b[100];
```

f)
```
typedef struct COMPLEX{float r,i;};
struct COMPLEX a,b[100];
```

g)
```
typedef struct COMPLEX{float r,i;};
COMPLEX a,b[100];
```

Referirea cîmpurilor unei structuri se face prin calificare, folosind operatorul . (punct), la fel ca în Pascal ***(A)***.

Exemple: `a.r` referă partea reală a variabilei *a*
`b[10].i` referă partea imaginară a elementului cu indicele 10

```
#include <string.h>
main()
{ struct articol {char nume[40];
                  char adresa[20];
                  int an, luna, zi;}
  struct articol pers;
   ……
  strcpy(pers.nume, "Popescu Ion");
  strcpy(pers.adresa, "Bucuresti, Pta. Romana 77");
  pers.an=1979; pers.luna=3; pers.zi=15;}
```

Exemplu de articol în articol:

```
typedef struct art_1 {int an, luna, zi;}
typedef struct articol {char nume[40];
                        char adresa[20];
                        art_1 data_nasterii; }
articol pers;
……
strcpy(pers.nume, "Popescu Ion");
strcpy(pers.adresa, "Bucuresti, Pta. Romana 6");
pers.data_nasterii.an=1979; pers.data_nasterii.luna=3;
pers.data_nasterii.zi=15;
```

Exemplu de masiv de articole:

```
struct articol {int cant;float pret;}
articol x[200];
……
x[7].cant=50; x[7].pret=100000;
```

x[0]		x[1]		x[2]		…	x[199]	
cant	pret	cant	pret	cant	pret	…	cant	pret

Exemplu de masive în articol:

```
struct articol {int cant[30];
                float pret[30];}
……
articol vanz;
……
vanz.cant[7]=50; vanz.pret[7]=100000;
```

cant					pret				
0	1	2	…	29	0	1	2	…	29

4. Operatori şi expresii

Expresiile sunt construcţii sintactice formate dintr-un operand sau mai mulţi operanzi legaţi prin operatori. Expresiile, într-un program, au o *valoare* şi un *tip* (spre deosebire de matematică, unde au doar valoare).

4.1 Operanzi

Un operand poate fi una din următoarele construcţii:

- un literal
- o constantă simbolică
- o variabilă simplă
- numele unui masiv
- numele unui tip de dată
- numele unei funcţii
- referirea la elementul unui tablou
- referirea la câmpul unei structuri
- apelul unei funcţii
- o expresie

Din ultima posibilitate rezultă că expresia este o construcţie recursivă.

Un operand are un tip şi o valoare. Valoarea se determină fie la compilare, fie la execuţie. Nu pot fi operanzi şirurile de caractere.

Exemple de operanzi: `4321`, `0xabc1` (literali);
Fie declaraţiile:

```
int a;
int v[100];
```

`a` şi `v` pot fi operanzi în expresii; *a* are tipul *int*, iar *v* are tipul *pointer la int*.

4.2 Operatori

Operatorii se clasifică după diferite criterii, precum: numărul operanzilor asupra cărora se aplică şi tipul de operaţie pe care o realizează. Majoritatea operatorilor sunt *unari* sau *binari*. Limbajul C utilizează şi un operator *ternar* (care se aplică asupra a trei operanzi). După tipul de operaţie realizată, operatorii sunt aritmetici, logici de relaţie etc.

Într-o expresie apar, de obicei, operatori din aceeaşi clasă, dar pot fi şi din clase diferite. Se pot scrie expresii complexe care să conţină operatori din toate clasele. La evaluarea acestor expresii se ţine cont de *priorităţile* operatorilor (numite şi clase de precedenţă), de *asociativitatea* lor şi *regula conversiilor implicite*.

4.2.1 Regula conversiilor implicite

Atunci cînd un operator binar se aplică la doi operanzi de tipuri diferite, înainte de a efectua operaţia, cei doi operanzi sunt aduşi la un tip comun. În general, operandul de tip *inferior* se converteşte către tipul operandului de tip *superior*.

În primul rînd se convertesc operanzii de tip *char* către tipul *int*.

Dacă operatorul se aplică la doi operanzi cu acelaşi tip, nu se face nici o conversie, rezultatul avînd acelaşi tip cu cei doi operanzi. Dacă valoarea lui depăşeşte domeniul asociat tipului de dată, rezultatul este eronat (eroarea de depăşire de domeniu).

Dacă operatorul se aplică la operanzi de tipuri diferite, se face conversie astfel:

1. dacă un operand este de tipul *long double*, celălalt se converteşte la acest tip, iar rezultatul va fi de tip *long double*,
2. altfel, dacă un operand este de tip *double*, celălalt se converteşte la acest tip, iar rezultatul va fi de tip *double*,
3. altfel, dacă un operand este de tip *float*, atunci celălalt se converteşte la acest tip, iar rezultatul va fi de tip *float*,
4. altfel, dacă un operand este de tip *unsigned long*, atunci celălalt se converteşte la acest tip, iar rezultatul va fi de tip *unsigned long*,
5. altfel, dacă un operand este de tip *long*, atunci celălalt se converteşte la acest tip, iar rezultatul va fi de tip *long*,
6. altfel, unul din operanzi trebuie să fie de tip *unsigned*, iar celălalt de tip *int*; acesta se converteşte la tipul *unsigned*, iar rezultatul va fi de tip *unsigned*.

Regula se aplică pe rînd pentru fiecare operator din cadrul unei expresii, obținînd în final tipul expresiei.

4.2.2 Operatori de atribuire

Spre deosebire de alte limbaje (de exemplu Pascal) în care atribuirea este executată de o *instrucțiune*, în C aceasta se realizează prin intermediul unei *expresii* cu operator de atribuire.

Forma generală este: `v=expresie`

unde *v* este o variabilă simplă, referință către un element al unui masiv sau un cîmp al unei structuri sau o expresie în urma evaluării căreia se obține o adresă. În C, entitatea care se poate afla în stînga unui operator de atribuire (care poate primi valoare) se numeşte *left value*.

Pentru a se putea efectua atribuirea, tipul expresiei şi tipul entității din stînga operatorului de atribuire trebuie să fie măcar compatibile. Dacă sînt incompatibile se produce eroare la compilare (deoarece compilatorul nu poate insera în codul obiect apelurile pentru operațiile de conversie).

Efectele atribuirii constau în:

- evaluarea expresiei din dreapta operatorului de atribuire, cu determinarea unei valori şi a unui tip;
- memorarea valorii expresiei din dreapta în variabila din stînga;
- întreaga expresie de atribuire are valoarea şi tipul variabilei din stînga.

Observație: În C, o instrucțiune simplă poate fi construită pe baza unei expresii. De exemplu:

`a=23` este o expresie de atribuire;

`a=23;` este o instrucțiune care provoacă evaluarea expresiei de atribuire;

`a<b` este o expresie;

`a<b;` este o instrucțiune care provoacă evaluarea expresiei de atribuire (rezultatul, în acest caz, nu are semnificație algoritmică, deoarece nu este folosit în nici un fel).

Deoarece atribuirea este o expresie, iar operandul din stînga este o expresie, în C se pot compune expresii de forma: `v1=(v=expresie)`. Întrucît operatorul de atribuire are prioritate mică, parantezele nu sînt necesare iar expresia devine `v1=v=expresie`. Evaluarea se face de la dreapta, aşa cum sugerează forma cu paranteze: se evaluează expresia `v=expresie`, în urma căreia `v` primeşte o valoare; aceasta este şi valoarea întregii expresii; această valoare este atribuită lui `v1`. În acest mod se pot construi expresii cu mai multe atribuiri succesive.

Dacă este nevoie, înaintea atribuirilor se fac conversii: valoarea care se atribuie (rezultată în urma evaluării expresiei) este convertită la tipul entității care primeşte valoarea. Ca urmare a acestor conversii pot să apară pierderi de informație sau chiar coruperi totale ale valorii expresiei.

Tabelul 4.1 Pierderi de informaţie în expresia de atribuire

Tip destinaţie	Tip expresie	Pierderi de informaţie
signed char	unsigned char	dacă valoarea este mai mare decît 127, destinaţia va fi un întreg negativ, rezultat imprevizibil
unsigned char	[short] int	octetul cel mai semnificativ, rezultat imprevizibil
unsigned char	long int	cei mai semnificativi trei octeţi, rezultat imprevizibil
[short] int	long int	cei mai semnificativi doi octeţi, rezultat imprevizibil
int	float	rezultat imprevizibil
float	double	se reduce precizia (cu rotunjire)
double	long double	se reduce precizia (cu rotunjire)

Rezultatele imprevizibile provin din faptul că nu se face conversie propriu-zisă, ci copiere binară din reprezentarea internă a rezultatului evaluării expresiei către destinaţie.

Exemplu:

```
int a;
char b;
float c;
b=a=c=180.5;
```

Lui *c* i se atribuie valoarea 180.5, lui *a* i se atribuie valoarea 180, iar lui *b* i se atribuie valoarea –76.

Exerciţiu:

```
#include <stdio.h>
main()
{ signed char a;
  unsigned char b=234;
  a=b;                          /* a= -22 */
  int c=435;
  b=c;                          /* b=179 */
  long int d=435675;
  b=d;                          /* b=219 */
  c=d;                          /* c=-23077 */
  float e=-23.123445;
  c=e;}                 /* c=-23 */
```

Temă: verificaţi exerciţiul anterior, prin includerea unor funcţii de afişare adecvate.

Operatorul `=` poate fi combinat cu alţi operatori, obţinînd operatori mai complecşi, care, pe lîngă atribuire, mai execută o operaţie. Forma generală a noilor operatori este `op=`, unde `op` este un operator binar, aritmetic sau logic pe biţi (`/`, `%`, `*`, `-`, `+`, `<<`, `>>`, `&`, `^`, `|`).

O expresie de forma

```
v op=expresie
```

este echivalentă cu

```
v=v op (expresie)
```

Efectele aplicării operatorului compus sînt: se evaluează *expresie*, se aplică operatorul *op* între valoarea lui *v* şi rezultatul obţinut, iar valoarea obţinută se atribuie lui *v*.
Exemplu: expresiile `i*=5` şi `i=i*5` sînt echivalente.

4.2.3 Operatori aritmetici

În ordinea priorităţii lor, operatorii aritmetici sînt:
i. operatorii unari `+`, `-`, `++`, `--`
ii. operatorii binari multiplicativi `*`, `/`, `%`
iii. operatorii binari aditivi `+`, `-`

Operaţiile efectuate de aceşti operatori sînt descrise în tabelul următor:

Tabelul 4.2 Operatorii aritmetici

Semnificaţie operaţie	Operator
Schimbare semn	-
Păstrare semn (nici un efect, nu este folosit)	+
Decrementare (post sau pre)	--
Incrementare (post sau pre)	++
Adunare	+
Scădere	-
Înmulţire	*
Împărţire	/
Împărţire întreagă (cît)	/
Împărţire întreagă (rest)	%

Observaţie: Cu excepţia operatorului %, care admite numai operanzi întregi, ceilalţi admit toate tipurile numerice.

Cîţiva dintre operatorii prezentaţi anterior sînt specifici limbajului C şi necesită unele precizări.

Operatorii `++` şi `--` admit operanzi de orice tip scalar. Efectul obţinut depinde de poziţia lor faţă de operand, astfel:
i. dacă apar înaintea operandului, valoarea acestuia este modificată înainte de a fi utilizată la evaluarea expresiei din care face parte (`++var` sau `--var`: preincrementare/predecrementare);
ii. dacă apar după operand, valoarea acestuia este folosită la evaluarea expresiei din care face parte, apoi este modificată (`var++` sau `var--`: postincrementare/ postdecrementare).

Incrementarea/decrementarea au ca efect modificarea valorii operandului cu o unitate. Semnificaţia unei unităţi depinde de tipul operandului asupra căruia se aplică. Pentru operanzi numerici, o unitate înseamnă unu.

Exemplu:

`y=x++;` este echivalent cu secvenţa `y=x; x=x+1;`
`y=--x` este echivalent cu secvenţa `x=x-1; y=x;`

Operatorul % are ca rol obţinerea restului unei împărţiri întregi. El este similar cu operatorul *mod* din Pascal.

Operatorul / are efect diferit, în funcţie de tipul operanzilor:

i. dacă cel puţin un operand este de tip real, se execută împărţire reală;

ii. dacă ambii operanzi sînt întregi, se execută împărţire întreagă şi se obţine cîtul.

În limbajul C, calculul cîtului şi restului unei împărţiri întregi se poate realiza şi prin intermediul funcţiei *div*. Rezultatul funcţiei are tipul *div_t*, definit în biblioteca *stdlib.h* astfel:

```
typedef struct {long int quot;  //cît
                long int rem;   //rest
                } div_t;
```

Exemplu: determinarea cîtului (*cit*) şi restului (*rest*) împărţirii numărului *m* la numărul *n* se realizează astfel:

```
div_t x;
int m,n,cit,rest;
x=div(m,n);
cit=x.quot;
rest=x.rem;
```

4.2.4 Operatori logici şi relaţionali

Operatorii logici sînt prezentaţi în tabelul 4.3 (în ordinea priorităţii), iar operatorii relaţionali în tabelul 4.4.

Tabelul 4.3 Operatorii logici

Semnificaţie operaţie	Operator
Negare	!
Şi logic	&&
Sau logic	||
Sau exclusiv	Nu există

În limbajul C nu există tipul de dată logic. Operanzii asupra cărora se aplică operatorii logici sînt convertiţi în valori numerice şi interpretaţi conform convenţiei: *adevărat* pentru valoare nenulă şi *fals* pentru zero. Rezultatul unei operaţii logice este de tip *int*, conform convenţiei: pentru *adevărat* valoarea 1, iar pentru *fals* valoarea 0.

La evaluarea expresiilor logice se aplică principiul evaluării parţiale: dacă expresia este de tip aditiv (operanzi legaţi prin operatorul *sau*), în momentul în care s-a stabilit că un operand are valoarea *adevărat* toată expresia va fi *adevărată* şi nu se mai evaluează restul operanzilor. Asemănător, pentru o expresie multiplicativă (operanzi legaţi prin operatorul *şi*), dacă un operand are valoarea *fals*, expresia are valoarea *fals* şi nu se mai evaluează restul operanzilor.

Operația *sau exclusiv* între doi operanzi *a* şi *b* se efectuează cu expresia:

```
!a&&b||!b&&a
```

a	b	!a	!b	!a&&b	!b&&a	!a&&b||!b&&a
0	0	1	1	0	0	0
diferit de 0	diferit de 0	0	0	0	0	0
diferit de 0	0	0	1	0	1	1
0	diferit de 0	1	0	1	1	1

Rezultatul unei operații relaționale este de tip *int*, conform convenției: pentru *adevărat* valoarea 1, iar pentru *fals* valoarea 0. Pentru a nu se greşi la evaluările unor expresii complexe, este indicat să se facă raționamentul bazat pe logica booleană (cu valori de *adevărat* şi *fals*).

Tabelul 4.4 Operatorii relaționali

Semnificație operație	Operator
Mai mare	>
Mai mare sau egal	>=
Mai mic	<
Mai mic sau egal	<=
Egal	==
Diferit	!=

Exemplu: expresia `a<0||b<0` are valoarea 1, dacă cel puțin unul din operanzii *a* şi *b* este negativ, respectiv valoarea 0 în caz contrar.

4.2.5 Operatori la nivel de bit

Limbajul C permite operații pe biți, ca şi Pascal. Operatorii folosiți sînt prezentați în tabelul 4.5.

Tabelul 4.5. Operații pe biți

Semnificație operație	Simbol operator
Şi logic pe biți	&
Sau logic pe biți	|
Sau exclusiv logic pe biți	^
Negare (complement față de 1)	~
Deplasare la dreapta	>>
Deplasare la stînga	<<

Operanzii pot fi de orice tip întreg. Dacă este nevoie, operanzii sînt extinşi la reprezentare pe 16 biţi. Execuţia are loc bit cu bit. Pentru o pereche de biţi (x,y), valorile posibile rezultate în urma aplicării operatorilor logici la nivel de bit sînt prezentate în tabelul 4.6.

Tabelul 4.6 Rezultatele operaţiilor logice pe biţi

x	y	x&y	x|y	x^y	~x
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Exemple:

```
char i=23,
j=0xf2; /* j are valoarea 242 */
i&j;    /* rezultatul este 18 */
i|j;    /* rezultatul este 247 */
i^j;    /* rezultatul este 229 */
~i;     /* rezultatul este -24 */
```

`~1234` se reprezintă binar 10011010010; se extinde la 16 biţi: 0000010011010010; apoi se aplică operatorul de complementare faţă de 1: 1111101100101101 (64301 dacă operandul era fără semn, -1235 dacă era cu semn).

`~-1234` se reprezintă binar 10011010010; se extinde la 16 biţi: 0000010011010010; se negativează: 1111101100101110; apoi se aplică operatorul de complementare faţă de 1: 0000010011010001 (1233).

`-~1234` se reprezintă binar 10011010010; se aplică operatorul de complementare faţă de 1: 1111101100101101; apoi se negativează: 10011010011 (1235).

Temă: verificaţi exemplele anterioare, prin includerea unor funcţii de afişare adecvate.

Operaţiile de deplasare au următoarea sintaxă:

`operand<<număr_poziţii` şi `operand>>număr_poziţii`

Operanzii sînt de tip întreg, iar *număr_poziţii* indică numărul de biţi cu care se deplasează valoarea. La deplasare se propagă valoarea 0, la stînga sau la dreapta.

Exemplu: `21<<1` are valoarea 42: 00010101<<1 = 00101010
`21>>2` are valoarea 5: 00010101>>2 = 00000101

Dacă tipul primului operand este *unsigned*, deplasarea spre stînga este echivalentă cu valoare=valoare*$2^{\text{număr_poziţii}}$, iar deplasarea spre dreapta este echivalentă cu valoare= [valoare/$2^{\text{număr_poziţii}}$], unde parantezele drepte semnifică *partea întreagă a cîtului*.

Operația de "mascare" (reținerea biților care interesează dintr-o valoare şi anularea celorlalți) este des întîlnită în aplicații. Reținerea valorii bitului din poziția *p* dintr-un număr *a* (se consideră cazul reprezentării pe un octet) se poate face prin deplasarea la stînga a lui *a* cu *p-1* poziții şi efectuarea operației & cu masca 0x1 (cu reprezentarea 00000001) sau, echivalent, prin "mascarea" lui *a* cu 2^{p-1} şi deplasare la stînga cu *p-1* poziții.

Exemple:
1. Următoarea secvență afişează, în ordine, biții unui octet (dată de tip char).

```
char c;
scanf("%d",&c);
for(int i=0;i<8;i++)
printf("%d",(c&((int)pow(2,7-i)))>>7-i);
```

2. Următoarea secvență afişează, în ordine inversă, biții unui octet (dată de tip char).

```
char c;
scanf("%d",&c);
for(int i=0;i<8;i++,c=c>>1)
printf("%d",c&1);
```

4.2.6 Operatorul "," (virgulă)

Operatorul "," este, de asemenea, specific limbajului C, nefiind implementat în Pascal. Pe lîngă rolul de separator într-o listă, virgula este considerată şi operator, într-o secvență de forma:

```
expresie_1, expresie_2, ..., expresie_n;
```

Operatorul "," induce evaluarea succesivă, de la stînga la dreapta, a expresiilor, întreaga secvență fiind tratată ca o expresie căreia i se atribuie în final valoarea şi tipul corespunzătoare ultimei expresii evaluate (*expresie_n*).

Acest operator se utilizează acolo unde este legal să apară o expresie în program şi dorim să realizăm un calcul complex, exprimat prin mai multe expresii.

Exemplu:

```
int a=10, b=3, c, d;
d=(c=a+b, b+=c, a/2);
```

Valorile variabilelor după evaluarea expresiei sînt: a = 10, c = 13, b = 16, d = 5.

Notă: În limbajul C, construcția care utilizează perechi de paranteze rotunde de tipul *(expresie)* este considerată o expresie, tipul ei depinzînd de regulile expresiei interioare.

4.2.7 Operatorul de conversie explicită

Asemănător limbajului Pascal, se poate converti explicit tipul unei expresii către un tip dorit de utilizator (dar compatibil). Expresia de conversie este asemănătoare cu cea din Pascal, doar sintaxa diferă minor (parantezele includ tipul spre care se converteşte, nu operandul care se converteşte, ca în Pascal). Forma generală este:

```
(tip)operand
```

Se observă că acest operator nu are un simbol explicit. El este format din numele unui tip de dată inclus între paranteze. Construcția este numită *expresie cast* (conversia explicită se numeşte *typecast*). Trebuie reținut că nu se schimbă efectiv tipul operandului, doar se foloseşte în expresie valoarea operandului convertită la un alt tip. Operația de conversie de tip nu are efect permanent.

Exemple:
```
int a=7;
float b=(float)a;
```
După executarea secvenței, *a* are valoarea 7 (nu 7.0) nefiind afectată în nici un fel de operația de conversie de tip. *B* are valoarea 7.0 (obținută prin conversia valorii lui *a* către tipul *float*).

```
int a=7;
float b=9.3;
int c;
c=a+int(b);
```
După executarea secvenței, *c* are valoarea 16, obținută prin adunarea valorii lui *a* (7) şi a celei rezultate prin conversia valorii lui *b* către tipul întreg (9).

4.2.8 Operatorul *dimensiune*

Operatorul *dimensiune* este folosit pentru a afla dimensiunea în octeți a spațiului de memorie ocupat de o variabilă de un tip oarecare. Simbolul său este *sizeof* şi poate fi folosit în una din formele:

```
sizeof var
sizeof(var)
sizeof(tip)
```
unde *tip* este numele unui tip de dată sau descrierea unui tip de dată, iar *var* poate fi numele unei variabile simple, numele unui masiv, numele unei structuri, referirea la un element al unui masiv sau la un cîmp al unei structuri.

În primele două forme, rezultatul va fi numărul de octeți alocați entității respective. Pentru ultima formă, rezultatul este numărul de octeți care se alocă pentru a reprezenta o variabilă de tipul *tip*.

Exemple:

```
int x;
sizeof(x)    are valoarea 2
sizeof(int) are valoarea 2

long double a[10];
sizeof(a[4]) are valoarea 10 (tipul long double se reprezintă pe 10 octeți)
sizeof(a)    are valoarea 100 (10 elemente de tip long double)
```

4.2.9 Operatorii paranteze

Parantezele au rolul de a include o expresie sau lista de parametri a funcțiilor. Prin includerea subexpresiilor între paranteze se modifică ordinea de evaluare a unei expresii.

Prin includerea unei expresii între paranteze se obține un operand, asupra căruia nu se pot aplica însă orice operatori; de exemplu, nu se pot aplica operatorii de incrementare/decrementare.

Exemplu: construcțiile `(i*5)++` sau `--(a+b)` sînt eronate.

La apelul unei funcții, parantezele rotunde sînt numite *operatori de apel de funcție*, ele delimitînd lista parametrilor reali.

Parantezele pătrate au rolul de a include expresii care reprezintă indici pentru accesarea elementelor unui masiv. Ele se numesc *operatori de indexare*.

4.2.10 Operatorul condițional

Operatorul condițional este singurul care are trei operanzi şi este specific limbajului C. Forma generală este:

```
expresie_1?expresie_2:expresie_3
```

unde *expresie_1*, *expresie_2* şi *expresie_3* sînt expresii. Operatorul condițional are simbolul `?:`. Cele două caractere care compun simbolul apar intercalate între cei 3 operanzi, conform formei generale. Construcția se numeşte expresie condițională. Valoarea şi tipul acestei expresii sînt identice cu ale lui *expresie_2* – dacă *expresie_1* este adevărată (nenulă) – sau cu ale lui *expresie_3* – dacă *expresie_1* este falsă (zero).

Exemplu:

```
int a=7,b=9,c;
c=(a>b)?a:b;
```

Variabila *c* primeşte valoarea maximă dintre *a* şi *b* (în exemplul de mai sus, 9).

4.2.11 Alți operatori

În limbajul C mai sînt disponibili următorii operatori:

i. operatorul de calificare (cu simbolul . – caracterul punct), folosit pentru a accesa un cîmp al unei structuri, ca în Pascal;

ii. operatorul de calificare (cu simbolul ->) folosit pentru a accesa un cîmp atunci cînd se ştie adresa unei structuri;

iii. operatorul de referențiere (cu simbolul &) folosit pentru a extrage adresa unei variabile;

iv. operatorul de referențiere (cu simbolul *) folosit pentru a defini un pointer;

v. operatorul de dereferențiere (cu simbolul *) folosit pentru a extrage valoarea de la o anumită adresă.

Ultimii operatori vor fi studiați în capitolul destinat lucrului cu adrese.

4.2.12 Evaluarea expresiilor

Expresiile sînt evaluate pe baza următoarelor reguli:

- *Precedența*: determină ordinea de efectuare a operațiilor într-o expresie în care intervin mai mulți operatori (gradul de prioritate);
- *Asociativitatea*: indică ordinea de efectuare a operațiilor în cazul unui set de operatori cu aceeaşi precedență;
- *Regulile de conversie de tip*: asigură stabilirea unui tip comun pentru ambii operanzi, pentru fiecare operație care solicită acest lucru şi în care tipurile diferă.

Asociativitatea şi precedența operatorilor (începînd cu prioritatea maximă) sînt redate în tabelul 4.7.

Tabelul 4.7 Precedența operatorilor

Operatori	Asociativitate	Grad de prioritate
`() [] . ->`	de la stînga la dreapta	maxim
`+ - & * (unari) ++ -- (tip) sizeof ! ~`	de la dreapta la stînga	↓
`* (binar) / %`	de la stînga la dreapta	
`+ - (binari)`		
`<< >>`		
`< <= > >=`		
`== !=`		
`& (binar)`		
`^`		
`|`		
`&&`		
`||`		
`?:`		
`= <<= >>= += -= *= /= %= &= ^= |=`	de la dreapta la stînga	
`,`	de la stînga la dreapta	minimă

5. Operaţii de intrare/ieşire cu tastatura/monitorul

Deoarece tastatura şi monitorul sînt dispozitive asimilate fişierelor ASCII, operaţiile de intrare/ieşire se realizează cu **conversie** între reprezentarea internă a datelor (binară: virgulă fixă, virgulă mobilă etc.) şi cea externă (ASCII). Mecanismul operaţiei de scriere este prezentat în figura 5.1, iar al operaţiei de citire în figura 5.2.

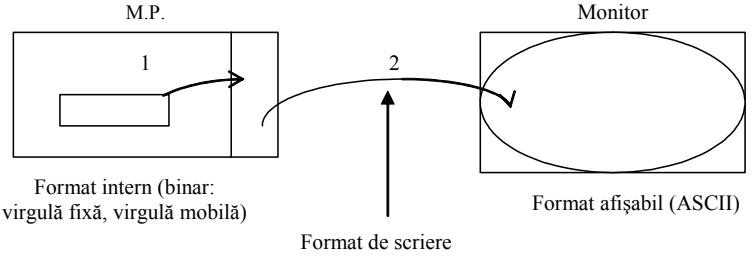


Figura 5.1 – Mecanismul operaţiei de scriere

Scrierea pe ecran se desfăşoară în doi paşi: datele din memoria principală (MP) sînt convertite în formatul extern de reprezentare (ASCII) şi transferate într-o zonă tampon (pasul 1). De aici sînt preluate şi afişate pe ecran (pasul 2).

La ***citire*** datele sînt preluate de la tastatură şi depuse într-o zona tampon (buffer), în format ASCII (pasul 1). De aici se preiau datele necesare, se convertesc în formatul intern şi se depun în memoria principală, în zonele corespunzătoare parametrilor funcţiilor de citire.

Se observă că operaţia de conversie are loc numai la transferul între zona tampon şi memoria principală. În general, în zona tampon datele sînt păstrate în acelaşi format cu dispozitivul asociat.

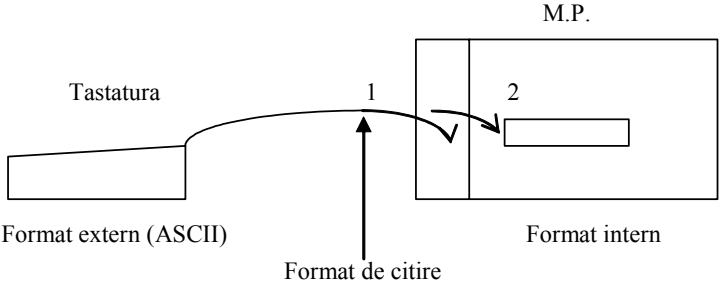


Figura 5.2 – Mecanismul operaţiei de citire

Datorită orientării spre portabilitate, în C nu au fost implementate instrucţiuni de intrare/ieşire. Operaţiile de intrare/ieşire se realizează prin apelul unor funcţii de bibliotecă ***(A)***.

Cele mai des utilizate funcţii de bibliotecă pentru afişarea datelor sînt: *printf*, *puts*, *putch*. La acestea se adaugă macroul *putchar*.

Cele mai des folosite funcţii de bibliotecă pentru citirea datelor de la tastatură sînt: *scanf*, *gets*, *getch*, *getche*. La acestea se adaugă macroul *getchar*.

Funcţiile *printf* şi *scanf* sînt numite funcţii de intrare/ieşire cu format (folosesc descriptori de format). Pentru utilizarea acestora, trebuie incluse explicit în programul sursă *head-ere*, prin directiva *#include*:

```
#include <stdio.h> sau
#include <conio.h>
```

Utilizatorul îşi poate defini propriile funcţii de intrare/ieşire dacă cele predefinite nu corespund necesităţilor.

5.1 Descriptori de format

Funcţiile de intrare/ieşire cu format au următoarea structură a parametrilor:

```
(<sir descriptor>, <listă de intrare/ieşire>)
```

Fiecărui element din lista de intrare/ieşire a operaţiei de citire/scriere trebuie să-i corespundă un descriptor de format în şirul descriptor. Descriptorii de format se pun în corespondenţă cu elementele listei de intrare/ieşire de la stînga la dreapta.

Exemplu:
```
int a,b;
printf("nr1=%i, nr2=%i ",a,b);
```

În exemplul precedent descriptorii de format sînt precizaţi prin `%i`. Primului `%i` îi corespunde *a*, celui de al doilea îi corespunde *b*.

Şirul de format este încadrat între ghilimele (este şir de caractere) şi conţine descriptorii de format. În cazul funcţiilor de afişare, descriptorii de format pot fi intercalaţi într-un text care va fi afişat. Fiecărui descriptor de format trebuie să-i corespundă exact un element din lista de intrare/ieşire. Formatele de intrare/ieşire îndeplinesc simultan rolurile de:

a) *Şabloane* ale modului de reprezentare externă a datelor:

- În Pascal, la citire este format implicit, iar la scriere poate fi format implicit sau explicit.
- În C, formatul explicit de scriere este mult mai dezvoltat, oferind programatorilor un control mai bun al modului de afişare.

b) *Parametri* de indicare a tipului de conversie. Forma generală a unui descriptor este:

```
%[cadraj][lăţime[.precizie]]cod_conversie
```

Se observă că, faţă de Pascal, un descriptor oferă, în plus, posibilitatea cadrării şi alegerea explicită a reprezentării externe a datei. În Pascal descriptorul de format are forma *[:lăţime[:precizie]]*; cadrarea se face implicit la dreapta, iar modul de reprezentare externă este ales automat de compilator, în funcţie de tipul datei care se afişează. În plus, în C, toţi descriptorii de format formează un parametru separat al funcţiei de intrare/ieşire.

Pentru scrierea datelor pe ecran se pot folosi următorii parametri în cadrul descriptorilor de format:

- *cadraj*: implicit datele se aliniază la dreapta cîmpului în care se scriu; prezenţa unui caracter minus determină alinierea datelor la stînga, iar un caracter plus determină alinierea la dreapta;
- *lăţime*: un număr care specifică lăţimea cîmpului, în număr de caractere, în care se va scrie valoarea. Dacă valoarea necesită mai multe caractere decît lăţimea dată explicit, afişarea se va face pe atîtea caractere cîte sînt necesare. Dacă lăţimea necesară este mai mică decît cea explicită, se completează cu caractere nesemnificative la stînga sau la dreapta, conform cadrajului specificat. Implicit, caracterele nesemnificative sînt spaţii. Dacă numărul care precizează lăţimea începe cu zero, caracterele nesemnificative sînt zerouri;
- *precizie*: indică precizia cu care se va scrie data. Dacă aceasta este o valoare reală, parametrul indică numărul de zecimale care vor fi afişate. Dacă partea fracţionară conţine mai multe cifre semnificative decît precizia, se rotunjeşte la ultima zecimală. Dacă data este un şir de caractere, indică numărul maxim de caractere care se vor scrie, indiferent de lungimea şirului;
- *cod_conversie*: este format din unul sau două caractere, cu semnificaţia precizată în tabelul 5.1.

Tabelul 5.1 Semnificaţia codurilor de conversie

Cod conversie	Semnificaţie	Observaţii
d	zecimal cu semn (int)	întregi
i	zecimal cu semn (int)	întregi
o	octal fără semn	întregi
u	zecimal fără semn	întregi
x	printf: hexazecimal fără semn (int) litere mici scanf: hexazecimal (int)	întregi
X	printf: hexazecimal fără semn (int) litere mari scanf: hexazecimal (int)	întregi
f	virgulă mobilă [-]dddd.ddd (format matematic)	virgulă mobilă
e	virgulă mobilă [-]d.ddd.e[+/-]dd (format ştiinţific)	virgulă mobilă
g	ca şi f sau e, depinde de precizie	virgulă mobilă
E	ca şi e, utilizează litera E pentru exponent	virgulă mobilă
G	ca şi g, utilizează litera E pentru exponent	virgulă mobilă
c	imprimă un singur caracter	întregi
s	imprimă caractere pînă la '\0' (NULL) sau pînă la .prec	şiruri
l sau L	poate să preceadă descriptorii d, o, x sau u, caz în care se face conversia din întreg spre long; dacă precede descriptorul f, se face conversie din float spre double	întregi
%	caracterul %	%% => %

Faţă de Pascal, limbajul C permite folosirea unor descriptori de format şi la citirea datelor. Pentru citirea datelor de la tastatură descriptorul de format poate conţine următorii parametri opţionali:

- *lăţime*: un număr care specifică lăţimea maximă a cîmpului din care se va citi valoarea respectivă;
- *un caracter asterisc* (se va citi de la tastatură valoarea respectivă, dar nu va fi atribuită niciunei variabile).

Dacă tipul obiectului nu concordă cu specificatorul de format (codul), se obţin rezultate imprevizibile (compilatorul nu controlează corespondenţa şi încearcă să facă reprezentarea conform descriptorului primit).

5.2. Funcţii de scriere/citire cu format

Pentru operaţiile de citire/scriere cu format, se folosesc funcţiile:

```
printf(<sir_descriptor>,<expresie1>,<expresie2> ...);
scanf(<sir_descriptor >,<adresa1>,<adresa2> ...);
```

Ambele funcţii sînt definite în biblioteca standard *stdio.h*.

Apelul *printf* transferă valori care vor fi afişate, iar apelul *scanf* transferă adrese ale variabilelor ale căror valori se vor citi. De aceea, la apelul funcţiei *printf*, parametrii reali pot fi variabile, constante, expresii sau apeluri de funcţii care returnează valori, în timp ce la apelul funcţiei *scanf*, parametrii reali trebuie să fie adresele variabilelor ale căror valori se citesc (adresele unde se depun valorile citite de la tastatură – subiectul va fi dezvoltat în capitolul *Subprograme*).

Deoarece limbajul C permite transferul parametrilor numai prin valoare, în cazul funcţiei *scanf* trebuie transmise explicit adresele variabilelor. În acest scop se foloseşte operatorul *&*, care ”extrage” adresa unei variabile.

5.2.1 Funcţia *printf*

Funcţia returnează numărul de octeţi (caractere) afişate în caz de succes sau -1 în caz de eroare. Parametrul <*sir_descriptor*> al funcţiei *printf* poate conţine caractere care se afişează ca atare şi coduri de format care definesc conversiile care se aplică asupra obiectelor precizate. Ca şi procedura *read* din Pascal, *printf* poate fi folosită pentru a afişa una sau mai multe date. Spre deosebire de procedura Pascal, primul parametru al funcţiei *printf* are semnificaţie prestabilită: este un şir de caractere care conţine descriptorii de format pentru toate datele care urmează a fi afişate – următorii parametri – în timp ce în Pascal formatul de afişare se specifică separat pentru fiecare dată. Deoarece funcţia returnează numărul de octeţi afişat, se pot face verificări suplimentare asupra rezultatelor operaţiei de afişare. Funcţia *printf* nu poate fi folosită pentru scrierea datelor în fişiere. În acest scop se poate folosi funcţia *fprintf*.

Exemple:

1.
```
printf("abcde");
printf("%s","abcde");
```
Ambele apeluri au acelaşi efect şi anume afişarea şirului de caractere *abcde*.

2.
```
printf("#%4c#", 'A');
```
Apelul are ca efect afişarea şirului `#   A#`.

3.
```
printf("#%-4c#", 'A');
```
Apelul are ca efect afişarea şirului `#A   #`.

4. Apelurile
```
printf("#%10s#","abcde");
printf("#%-10s#","abcde");
printf("#%15.10s#","Limbajul C++");
```
au ca efect afişarea pe ecran a şirurilor:
```
#     abcde#
#abcde     #
#     Limbajul C#
```

5.
```
printf("#%010d#",12345);
```
Apelul are ca efect afişarea şirului `#0000012345#`.

6.
```
printf("#%10o",123);
printf("#%10x",123);
```

Apelurile au ca efect afişarea şirurilor `# 173#` , respectiv `# 7b#` .

În şirul care defineşte formatul pot fi incluse *secvenţe escape* constituite din caracterul *backslash* şi un alt caracter. Sînt folosite următoarele combinaţii uzuale:

- **\n** (newline): determină trecerea cursorului la începutul rîndului următor al ecranului;
- **\t** (tab orizontal): determină afişarea unui caracter *tab* (implicit 8 spaţii pe ecran);
- **\a** (bell): determină declanşarea difuzorului intern pentru un sunet de durată şi frecvenţă standard;
- **\v** (tab vertical): determină saltul cursorului cu 8 rînduri mai jos;
- **\b** (backspace): revenirea cu o poziţie înapoi;
- **\r** (carriage return): revenire la începutul rîndului curent.

Includerea caracterului *newline* la sfîrşitul formatului în apelul funcţiei *printf* o face echivalentă cu procedura *write**ln*** din Pascal.

5.2.2 Funcţia *scanf*

Funcţia returnează numărul de cîmpuri corect procesate. Parametrul <*sir_descriptor*> poate conţine, în afara codurilor de conversie, caractere care vor trebui introduse ca atare de la tastatură în cadrul cîmpului respectiv.

Exemplu: dacă şirul descriptor conţine secvenţa `x=%d`, atunci de la tastatură trebuie introdus `x=<valoare>`.

Caracterele „albe” (spaţiu, tab) din cadrul şirului descriptor se ignoră. Cîmpul controlat de un specificator de format începe cu primul caracter curent care nu este „alb” şi se termină în unul din cazurile următoare:

- la caracterul după care urmează un caracter „alb”;
- la caracterul după care urmează un caracter care nu corespunde specificatorului de format care controlează cîmpul respectiv (de exemplu litere pentru tipul numeric);
- la caracterul prin care se ajunge la lungimea maximă a cîmpului, dacă aceasta a fost specificată.

În cazul citirii de caractere nu se aplică această regulă, cîmpul conţinînd un caracter oarecare. În cazul prezenţei unui caracter * în specificatorul de format, valoarea introdusă de la tastatură nu se atribuie nici unei variabile şi nu îi va corespunde un element din lista de intrare/ieşire.

Citirea se întrerupe, în cazul unei erori, în locul în care s-a produs ea. Eroarea poate fi cauzată de:

- necorespondenţa între textul curent introdus şi specificatorul de format care controlează cîmpul respectiv;
- neconcordanţa între data din cîmp şi specificatorul de format sub controlul căruia se face citirea.

La citire, datele externe pot fi:

- separate prin spațiu, tab, CR/LF sau virgulă;
- texte externe cu lungimi prestabilite (se precizează lungimea textului extern).

Exemple:

```
int x,y;
scanf("%i %i", &x,&y);  => separate prin spațiu
scanf("%i, %i", &x,&y); => separate prin virgulă
scanf("%5s%2s",x,y);    => x este şir de 5 caractere, y este şir de 2 caractere
```

Deoarece masivele de date sînt pointeri, atunci cînd este posibilă specificarea numelui, nu mai este nevoie de operatorul &.

În continuare se prezintă cîteva exemple de citire a unor texte.

Exemple:

```
char den[20];
printf("nume: ");
scanf("%s", den);   /* den este pointer */
```

Dacă se citeşte un element al vectorului *den*, trebuie procedat astfel:

```
char den[20];
printf("caracter: ");
scanf("%c", &den[0]);/* se scrie &den[0] deoarece den[0] nu este
                        pointer*/
```

Observație: la citire, şirul introdus se încheie prin tastele <*Enter*> sau *spațiu* (chiar dacă se precizează lungimea cîmpului).

Exemple:

1.
```
char s1[20],s2[20];
scanf("%s %s", s1,s2);
```

Dacă de la tastatură se introduce şirul *Limbaje evoluate*, atunci cele două variabile vor conține: *s1* valoarea "Limbaje" şi *s2* valoarea "evoluate".

2.
```
char s1[20],s2[20];
scanf("%2s %8s",s1,s2);
```

Introducînd de la tastatură şirul de la exemplul 1, cele două variabile vor conține: *s1* valoarea „Li", deoarece se citesc maxim două caractere şi *s2* valoarea „mbaje", deoarece cîmpul se termină la întîlnirea caracterului spațiu.

Se observă că, spre deosebire de procedurile de citire din Pascal, în C se pot preciza lungimile cîmpurilor externe din care se citesc datele, prin însăşi sintaxa funcțiilor. Folosind valoarea returnată de funcție, se pot face validări asupra citirii corecte a datelor. O altă deosebire constă în faptul că *scanf* nu poate fi folosită pentru citirea de date din fişiere. În acest scop se foloseşte funcția *fscanf*.

5.3 Funcţii de intrare/ieşire fără format

5.3.1 Funcţiile *gets* şi *puts*

Limbajul C are funcţii de intrare/ieşire dedicate lucrului cu şiruri de caractere. Funcţia *gets* se apelează astfel:

```
gets(masiv_de_caractere);
```

unde *masiv_de_caractere* este adresa unde se va depune şirul de caractere citit (numele vectorului de caractere).

Funcţia *gets* este folosită pentru introducerea de la tastatură a unui şir de caractere. Spre deosebire de citirea cu *scanf* (cu descriptorul *%s*), şirul citit poate conţine şi spaţii, deoarece citirea se termină la întîlnirea caracterului CR/LF sau CTRL/Z. Valoarea citită se memorează în masivul *masiv_de_caractere* şi se returnează adresa unde a fost depus acest şir (adică exact adresa primită ca parametru). Dacă se introduce CR/LF fără a se introduce alte caractere se consideră şirul vid (în memorie, la adresa şirului se va găsi caracterul '\0'). Dacă se introduce numai CTRL/Z nu se citeşte nici un şir şi se returnează zero, adică pointer neiniţializat. Caracterul CR/LF nu se păstrează în şirul citit, fiind înlocuit cu '\0'.

Funcţia *puts* se apelează astfel: *puts(sir)*, unde *sir* este o expresie de tip *pointer spre caracter*. Funcţia *puts* este folosită pentru afişarea unui şir de caractere pe ecran. Ea returnează codul ASCII al ultimului caracter afişat, în caz de succes, sau –1 în caz de eroare. După afişarea ultimului caracter, cursorul trece la începutul rîndului următor. Caracterul NUL ('\0') nu este afişat, fiind înlocuit cu secvenţa CR/LF. Funcţia afişează, începînd de la adresa primită ca parametru, toate caracterele pînă la întîlnirea caracterului NUL ('\0'). Dacă la adresa primită ca parametru nu se află un şir terminat cu caracterul NUL, rezultatele sînt imprevizibile.

Funcţiile *gets* şi *puts* sînt definite în biblioteca *stdio.h*.

Exemple:

```
char sir[80];
printf("Introduceti un text:");
gets(sir);
printf("Sirul introdus este: %s\n", sir);

char sir[] = "Limbajul C se invata mai usor cind stii
Pascal\n";
puts(sir);
```

5.3.2 Funcţiile *getch* şi *getche*

Ambele funcţii sînt destinate citirii unui caracter de la tastatură. Ele nu au parametri şi returnează codul ASCII al caracterului citit. Deosebirea dintre ele constă în faptul că citirea cu *getch* se realizează fără ecou pe ecran (caracterul corespunzător tastei apăsate nu este afişat pe ecran). La apelul funcţiei *getche*, pe ecran apare caracterul corespunzător tastei apăsate.

În cazul apăsării unei taste corespunzătoare tastaturii centrale, cele două funcţii returnează codul ASCII asociat caracterelor direct afişabile. În cazul apăsării unei taste funcţionale sau a combinaţiilor cu tasta CTRL, care returnează o secvenţă de două caractere ASCII, funcţiile trebuie apelate de două ori: prima dată se returnează codul *null*, iar a doua oară se returnează codul ASCII al tastei apăsate. La apăsarea tastei *Enter* se returnează valoarea 13.

Funcţia *getch* se comportă identic cu funcţia *readkey* din Pascal. Singura diferenţă care apare în cazul funcţiei *getche* este ecoul pe ecran. Cele două funcţii sînt definite în biblioteca standard *conio.h*.

Exemplu: programul următor afişează codurile tastelor apăsate, pînă cînd se apasă tasta *Enter*.

```
#include<stdio.h>
#include<conio.h>
void main()
{ unsigned char c,c1;
  do {printf("\n"); c=getch();
      printf("\t >> %d",c);
      if(!c) {c1=getch(); printf(", %d",c1);}
    }
 while(c!=13);
 getch();}
```

Rezultatul apăsării şirului de taste `<Esc> F1 F12 a A <săgeată sus> <săgeată jos> <săgeată stînga> <săgeată dreapta> <page up> <page down> <Enter>` este următorul (pe fiecare rînd a fost afişată secvenţa codurilor corespunzătoare combinaţiei de taste):

```
>> 27
>> 0, 59
>> 0, 134
>> 97
>> 65
>> 0, 72
>> 0, 80
>> 0, 75
>> 0, 77
>> 0, 73
>> 0, 81
>> 13
```

5.3.3 Funcția *putch*

Funcția are ca parametru o valoare întreagă şi are ca efect afişarea caracterului ASCII corespunzător (vezi exemplul următor). Funcția *putch* returnează codul ASCII al caracterului afişat.

Dacă parametrul are valori în intervalul 0÷31, se vor afişa caractere grafice nestandard, corespunzătoare caracterelor ASCII de control (cu excepția caracterelor *nul* şi *bell*, pentru care nu se afişează nimic, *backspace*, care va produce revenirea cu o poziție înapoi pe rîndul curent, LF (cod 10), care produce trecerea pe rîndul următor, în aceeaşi coloană şi CR (cod 13), care va produce revenirea la începutul rîndului curent).

Dacă parametrul are valori în intervalul 32÷127, se vor afişa caracterele corespunzătoare tastaturii centrale (caractere direct afişabile/imprimabile).

Dacă parametrul are valori în intervalul 128÷255, se vor afişa caracterele grafice standard.

Dacă parametrul are valoare în afara domeniului 0÷255, atunci se afişează caracterul corespunzător interpretării ca valoare în virgulă fixă aritmetică a primului octet din reprezentarea valorii respective (conform tipului ei) – octetul cel mai puțin semnificativ, deoarece în reprezentarea internă ordinea octeților este inversată. Se poate face calculul pentru a prevedea ce caracter va fi afişat. Pentru valori peste 255, calculul este simplu: dacă *v* valoarea primită, se afişează caracterul cu codul v%256.

Funcția *putch* este definită în biblioteca standard *conio.h*.

Exemple:

- pentru valoarea 2561 se va afişa caracterul cu codul 1 (2561%256=1);
- pentru valoarea –45 se afişează caracterul cu codul 211 (reprezentarea binară, în virgulă fixă algebrică, a lui –45 este 11010011, un octet; interpretarea în virgulă fixă aritmetică este 211);
- pentru valoarea –1234 se afişează caracterul cu codul 46 (reprezentarea binară, în virgulă fixă algebrică, a lui –1234 este 11111011. 00101110, doi octeți – punctul a fost pus din motive de lizibilitate, pentru a separa octeții – el nu face parte din reprezentare; octetul cel mai nesemnificativ este 00101110; interpretarea în virgulă fixă aritmetică este 46);
- pentru valoarea –54321 se afişează caracterul cu codul 207 (reprezentarea binară, în virgulă fixă algebrică, a lui –54321 este 11111111.11111111.00101011. 11001111, patru octeți – punctele au fost puse din motive de lizibilitate, pentru a separa octeții – ele nu fac parte din reprezentare; octetul cel mai nesemnificativ este 11001111; interpretarea în virgulă fixă aritmetică este 207);
- Programul următor afişează caracterele codului ASCII (sau corespondentele lor grafice, pentru caractere de control şi neimprimabile) şi codurile lor. Pentru a evita suprascrierea unor valori, după caracterul cu codul 13 (revenire la începutul rîndului curent), s-a trecut în mod explicit pe rîndul următor, prin afişarea

caracterului *newline* (perechea CR/LF – codurile 10 şi 13). Dacă nu s-ar fi procedat aşa, al treilea rînd ar fi lipsit din rezultatul afişat de program. Din motive legate de aşezarea rezultatului în pagină, s-au afişat cîte 8 perechi cod/caracter pe un rînd.

```
#include<stdio.h>
#include<conio.h>
void main()
{ unsigned char c;
  int i,j;
  clrscr();
  for(i=0;i<32;i++)
    {for(j=0;j<8;j++)
      {if((i*8+j)==13)putch(10);
       printf("  %3d  ",i*8+j);
       putch(i*8+j);}
     printf("\n");}
  getch();}
```

Rezultatele execuţiei programului sînt:

```
  0       1  ☺    2  ☻    3  ♥    4  ♦    5  ♣    6  ♠    7
  8    9  ○   10
                      11  ♂   12  ♀   13
 14  ♬  15  ☼
 16  ►  17  ◄  18  ↕  19  ‼  20  ¶  21  §  22  ▬  23  ↨
 24  ↑  25  ↓  26  →  27  ←  28  ∟  29  ↔  30  ▲  31  ▼
 32     33  !  34  "  35  #  36  $  37  %  38  &  39  '
 40  (  41  )  42  *  43  +  44  ,  45  -  46  .  47  /
 48  0  49  1  50  2  51  3  52  4  53  5  54  6  55  7
 56  8  57  9  58  :  59  ;  60  <  61  =  62  >  63  ?
 64  @  65  A  66  B  67  C  68  D  69  E  70  F  71  G
 72  H  73  I  74  J  75  K  76  L  77  M  78  N  79  O
 80  P  81  Q  82  R  83  S  84  T  85  U  86  V  87  W
 88  X  89  Y  90  Z  91  [  92  \  93  ]  94  ^  95  _
 96  `  97  a  98  b  99  c 100  d 101  e 102  f 103  g
104  h 105  i 106  j 107  k 108  l 109  m 110  n 111  o
112  p 113  q 114  r 115  s 116  t 117  u 118  v 119  w
120  x 121  y 122  z 123  { 124  | 125  } 126  ~ 127  ⌂
128  Ç 129  ü 130  é 131  â 132  ä 133  à 134  å 135  ç
136  ê 137  ë 138  è 139  ï 140  î 141  ì 142  Ä 143  Å
144  É 145  æ 146  Æ 147  ô 148  ö 149  ò 150  û 151  ù
152  ÿ 153  Ö 154  Ü 155  ¢ 156  £ 157  ¥ 158  ₧ 159  ƒ
160  á 161  í 162  ó 163  ú 164  ñ 165  Ñ 166  ª 167  º
168  ¿ 169  ⌐ 170  ¬ 171  ½ 172  ¼ 173  ¡ 174  « 175  »
176  ░ 177  ▒ 178  ▓ 179  │ 180  ┤ 181  ╡ 182  ╢ 183  ╖
184  ╕ 185  ╣ 186  ║ 187  ╗ 188  ╝ 189  ╜ 190  ╛ 191  ┐
192  └ 193  ┴ 194  ┬ 195  ├ 196  ─ 197  ┼ 198  ╞ 199  ╟
200  ╚ 201  ╔ 202  ╩ 203  ╦ 204  ╠ 205  ═ 206  ╬ 207  ╧
208  ╨ 209  ╤ 210  ╥ 211  ╙ 212  ╘ 213  ╒ 214  ╓ 215  ╫
216  ╪ 217  ┘ 218  ┌ 219  █ 220  ▄ 221  ▌ 222  ▐ 223  ▀
224  α 225  ß 226  Γ 227  π 228  Σ 229  σ 230  µ 231  τ
232  Φ 233  Θ 234  Ω 235  δ 236  ∞ 237  φ 238  ε 239  ∩
240  ≡ 241  ± 242  ≥ 243  ≤ 244  ⌠ 245  ⌡ 246  ÷ 247  ≈
248  ° 249  · 250  · 251  √ 252  ⁿ 253  ² 254  ■ 255
```

Se observă pe rîndul al doilea al rezultatului, deplasarea cu o poziţie înapoi (practic pe ecran avem o deplasare aparentă de 2 poziţii: una prin revenire şi una prin neafişarea caracterului curent) prin afişarea caracterului *backspace* (cod 8) şi trecerea pe rîndul următor, fără a schimba coloana curentă la afişarea caracterului *line feed* (cod 10).

5.3.4 Macro-urile *getchar* şi *putchar*

Cele două macro-uri sînt definite în biblioteca *stdio.h* şi se apelează ca şi funcţiile *getch* şi *putch*. Diferenţa constă în modul de execuţie.

Macroul *getchar* permite citirea cu ecou pe ecran a caracterelor de la tastatură. Se pot citi numai caractere asociate codurilor ASCII direct afişabile, nu şi caracterele corespunzătoare tastelor speciale. Citirea nu se face direct de la tastatură. Toate caracterele tastate se introduc într-o zonă tampon pînă la apăsarea tastei *Enter*. În acest moment se introduce în zona tampon caracterul *newline* `'\n'` şi macro-ul returnează codul ASCII al caracterului curent din zona tampon. La următorul apel al lui *getchar* se returnează codul următorului caracter din zona tampon. Dacă la un apel al lui *getchar* nu mai sînt caractere disponibile în zona tampon, se introduc din nou de la tastatură caractere pînă la apăsarea tastei *Enter*.

La întîlnirea caracterului CTRL-Z (terminator de fişier), macro-ul returnează valoarea *EOF*, definită în *stdio.h*.

Macroul *putchar* are ca parametru o expresie al cărei rezultat reprezintă codul ASCII al caracterului care se doreşte să fie afişat. Se returnează codul ASCII al caracterului afişat sau –1 în caz de eroare.

Exemple:

```
#include<stdio.h>
#include<conio.h>
void main()
{ putchar(getchar()); putchar('\n');}
```

Programul aşteaptă introducerea de la tastatură a unui şir de caractere ASCII direct afişabile (orice altceva este ignorat) terminat cu *Enter*. Primul caracter din şirul introdus va fi preluat de macroul *getchar* care returnează codul ASCII al caracterului respectiv. Macroul *putchar* primeşte ca parametru codul ASCII returnat de *getchar* şi afişează caracterul corespunzător. Ca urmare, pe ecran se va afişa primul caracter care a fost introdus de la tastatură.

```
#include<stdio.h>
#include<conio.h>
void main()
{ char c;
  do
   putchar(c=getchar());
  while(c!=EOF);
  putchar('\n');}
```

La primul apel al lui *getchar*, *p*rogramul aşteaptă introducerea de la tastatură a unui şir de caractere ASCII direct afişabile (orice altceva este ignorat) terminat cu *Enter*. La fel ca înainte, primul caracter va fi afişat. La următoarele apeluri, se preia cîte un caracter din cele introduse anterior şi se afişează. Dacă în buffer nu mai sînt caractere, se aşteaptă din nou introducerea unui şir terminat cu *Enter*.

6. Realizarea structurilor fundamentale de control

Limbajele orientate spre programarea structurată cuprind instrucțiuni care implementează complet conceptele proiectării structurate şi modularizate a programelor. Instrucțiunile desemnează acțiuni care se aplică datelor în vederea obținerii rezultatelor scontate printr-un algoritm. Este recomandat ca studiul instrucțiunilor să se facă pornind de la structurile fundamentale definite de teoria programării structurate, pentru a fi scoase în evidență asemănările şi deosebirile față de limbajul cunoscut.

Un program trebuie să execute cel puțin o instrucțiune, chiar dacă aceasta este vidă. În limbajul C, un program se regăseşte sub forma unei funcții rădăcină, care, la limită, poate avea un corp vid:

```
void main() {}
```

Instrucțiunile se încheie cu caracterul; (punct şi virgulă – terminator de instrucțiune). După modul de realizare a construcțiilor sintactice şi al numărului de acțiuni descrise, există instrucțiuni simple şi instrucțiuni structurate.

De asemenea, pot fi create blocuri de instrucțiuni executabile, denumite instrucțiuni compuse. În tabelul 6.1 se redau elemente comparative între limbajele Pascal şi C.

Tabelul 6.1 Instrucțiunile în Pascal şi C

Pascal	**C**
O instrucțiune compusă este o secvență de instrucțiuni (simple, structurate sau compuse) delimitate de:	
cuvintele rezervate begin şi end	Caracterele { şi }
O instrucțiune simplă descrie o singură acțiune, unic determinată şi executată necondiționat:	
Atribuirea, goto, apelul procedurilor, inline, instrucțiunea vidă	goto, break, continue, instrucțiunea vidă, return
O instrucțiune structurată este o construcție care conține alte instrucțiuni (simple, structurate sau compuse), care se execută secvențial, alternativ sau repetitiv.	
Instrucțiunile sînt *separate* prin caracterul;. Există şi cuvinte rezervate care, pe lîngă rolul lor, îndeplinesc şi rol de ***separator*** (end, else, until), în prezența lor nefiind necesar sepa-ratorul ;.	Caracterul ; este ***terminator*** de instrucțiune şi este obligatoriu pentru a marca sfîrşitul fiecărei instrucțiuni.

Pe lîngă instrucţiunile care implementează conceptele programării structurate, C conţine şi instrucţiuni care contravin acestora, datorită orientării limbajului spre compactarea textului sursă şi uşurinţă în programare.

6.1 Instrucţiuni simple

6.1.1 Instrucţiunea vidă

Instrucţiunea vidă descrie acţiunea vidă. În C nu are o mnemonică explicită ***(A)***, fiind dedusă din contextul unor construcţii sintactice. Ea se foloseşte acolo unde trebuie să apară o instrucţiune, dar nu trebuie să se execute nimic. Situaţia este întîlnită de obicei în cazul instrucţiunilor structurate.

Exemple:
```
if (c==1); else c=2;
if (c==1); else;
{;}
```

6.1.2 Instrucţiunea de tip expresie

Instrucţiunea de tip expresie evaluează o expresie care, în cele mai dese cazuri, este de atribuire sau de apel al unei funcţii (vezi şi capitolul *Operatori şi expresii*). Instrucţiunea de tip expresie se obţine scriind terminatorul de instrucţiune după o expresie (acolo unde este legal să apară o instrucţiune în program). Forma generală este:

```
expresie;
```

Exemple:
```
int x;
x=10;
x=x+3; x+=4;
++x;            // se incrementează x
putch('\n');  // instructiune de apel al functiei putch
```

6.2 Instrucţiunea compusă

Instrucţiunea compusă este o succesiunea de instrucţiuni şi declaraţii, cuprinse între o pereche de acolade. Se preferă ca declaraţiile să fie plasate înaintea instrucţiunilor. Forma generală este:

```
{declaratii
 instructiuni}
```

Declaraţiile sînt valabile în interiorul instrucţiunii compuse.

Exemplu:
```
int a,b;
{int t;
 t=a;
 a=b;
 b=t;}
```

Variabila *t* este definită în momentul în care se ajunge la prima instrucțiune din cadrul instrucțiunii compuse (*t=a*) şi nu mai este definită după execuția ultimei instrucțiuni din bloc.

Instrucțiunea compusă se utilizează acolo unde este nevoie de o instrucțiune dar sînt necesare acțiuni complexe, care nu pot fi exprimate printr-o instrucțiune simplă. Situația este întîlnită în cadrul structurilor alternative şi repetitive.

6.3 Instrucțiuni structurate

În continuare, prin instrucțiune se înțelege o instrucțiune simplă, structurată sau compusă.

6.3.1 Realizarea structurilor alternative

a) *Structura alternativă simplă* permite realizarea unei ramificări logice binare, în funcție de valoarea unei condiții (în C, expresie cu rezultat logic). Instrucțiunea care realizează această structură este *if* ***(A)***:

```
if(expresie)instrucțiune_1;
[else instrucțiune_2];
```

Expresia poate fi de orice tip. Dacă valoarea expresiei este diferită de zero (valoare asociată din punct de vedere logic cu *adevărat*) se execută *instrucțiune_1*; în caz contrar se execută *instrucțiune_2* sau se iese din structură (cînd construcția *else* lipseşte).

Exemple:

1)
```
if(a>b) a=c;
else a=b;
```

2)
```
if(a==3)
  if(b==4) c=2;
  else c=1;
else c=0;
```

3)
```
if(c!=) a=b;
```

Observații:

- Terminatorul de instrucțiuni `;` se scrie obligatoriu înainte de *else* ***(D)***, care nu mai are rol de separator cum avea în Pascal (unde înlocuia separatorul `;`). Rolul lui este de a termina instrucțiunea de pe ramura directă a instrucțiunii *if*. La limită aceasta poate fi instrucțiunea vidă.
- Expresia instrucțiunii *if* este întotdeauna inclusă între paranteze ***(D)***, acest lucru fiind valabil pentru toate instrucțiunile condiționale din C.

În locul unei structuri *if-then-else* se poate utiliza operatorul condițional `?:`, atunci cînd *instrucțiune_1* şi *instrucțiune_2* sînt de tip expresie.

Exemple:

1. Determinarea maximului dintre două numere:

```
max=(a>b) ? a:b;
```

2. Rezolvarea ecuaţiei de gradul I, ax+b=0:

```
a ? printf("Solutia este %10.5f",(float)-b/a) : b ? printf("Ecuatia
nu are solutii") : printf("Solutii: orice x real");
```

b) *Structura alternativă multiplă* (sau *selectivă*) permite alegerea unei acţiuni dintr-un grup, în funcţie de valorile pe care le poate lua un selector (mecanismul este redat în figura 6.1). În limbajul C structura se simulează cu instrucţiunea *switch* (echivalentă instrucţiunii *CASE* din Pascal), a cărei sintaxă este:

```
switch(expresie)
 {case c_1:  instrucţiune_1;
  case c_2:  instrucţiune_2;
    ……
  case c_n:  instrucţiune_n;
  [default:  instrucţiune;]}
```

unde: e*xpresie* este de tip ordinal (care se asociază tipului întreg); *c_1, c_2, ..., c_n* sînt expresii constante, de tip *int*, unice (o valoare nu poate să apară de două sau mai multe ori); *instrucţiune_1*, *instrucţiune_2*, ..., *instrucţiune_n*, *instrucţiune* sînt instrucţiuni (simple, compuse sau structurate). Dacă sînt instrucţiuni compuse, nu este nevoie să fie incluse între acolade.

Instrucţiunea *switch* evaluează expresia dintre paranteze, după care caută în lista de expresii constante acea valoare obţinută. Dacă este găsită, se execută instrucţiunea asociată valorii respective şi, secvenţial, toate instrucţiunile care urmează, pînă la terminarea structurii de selecţie. Dacă valoarea căutată nu este găsită în listă şi ramura *default* este prezentă, se execută instrucţiunea asociată acesteia. Dacă ramura *default* lipseşte nu se execută nimic. Pentru a limita acţiunea strict la execuţia instrucţiunii asociate unei valori, instrucţiunea asociată acesteia trebuie să fie compusă şi ultima instrucţiune simplă din componenţa ei trebuie să fie *break* (execuţia instrucţiunii *break* determină ieşirea din structura *switch*).

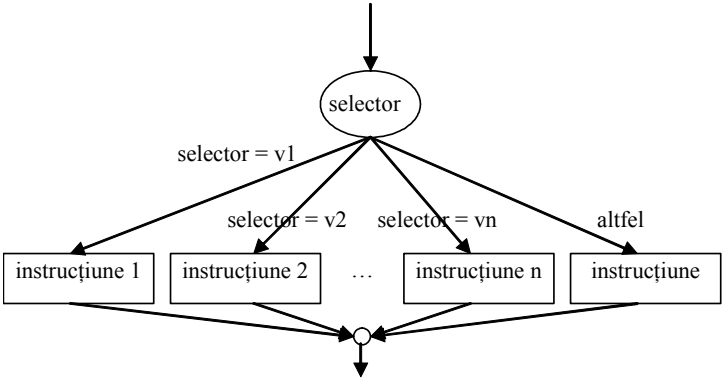


Figura 6.1 - Structura alternativă multiplă

Exemplu: dacă într-un program C este prezentă o instrucțiune *switch* de forma:

```
switch (expresie)
 {case c_1: instrucțiune_1;
  case c_2: instrucțiune_2;}
```

atunci, ea se execută astfel:

- dacă valoarea expresiei este *c_1*, se execută *instrucțiune_1* şi apoi *instrucțiune_2*, dacă *instrucțiune_1* nu defineşte ea însăşi o altă secvență de continuare;
- dacă valoarea expresiei este *c_2*, se execută *instrucțiune_2*;
- dacă valoarea expresiei diferă de *c_1* şi *c_2*, nu se execută nimic.

Exemple:

```
char litera;
printf("\nTastati o litera intre a si c");
scanf("%c",&litera);
switch (litera)
 {case 'a':
  case 'A': printf("litera a sau A");break;
  case 'b':
  case 'B': printf("litera b sau B");break;
  case 'c':
  case 'C': printf("litera c sau C");break;
  default: printf("alta litera");}
```

Se observă că instrucțiunea *switch* nu realizează natural structura CASE-OF (ieşirea unică trebuie să fie asigurată prin instrucțiunea *break*). Pentru a executa aceeaşi instrucțiune pentru mai multe valori se procedează ca mai sus: valorile respective se înscriu pe ramuri succesive, asociind instrucțiunea dorită ultimei ramuri; restul ramurilor nu vor avea instrucțiuni asociate ***(D)***. Pentru acest scop, limbajul Pascal permitea scrierea mai multor valori pe aceeaşi ramură.

Pentru a simula în C structura fundamentală CASE-OF, fiecare ramură trebuie să se încheie cu *break*.

6.3.2 Realizarea structurilor repetitive

a) *Structura repetitivă condiționată anterior* este implementată prin instrucțiunea *while* ***(A)*** cu forma generală:

```
while (expresie) instrucțiune;
```

Instrucțiunea *while* se execută astfel: se evaluează expresia şi dacă este adevărată (diferită de zero) se execută instrucțiunea (simplă, compusă sau structurată); se reia procesul pînă cînd la evaluarea expresiei se obține valoarea zero. În acest moment se încheie execuția instrucțiunii *while*. Pentru a asigura ieşirea din ciclu, instrucțiunea trebuie să afecteze valoarea expresiei. Dacă la prima evaluare a expresiei se obține valoarea zero, atunci nu se execută nimic. Instrucțiunea *while* din C (ca şi din Pascal) implementează natural structura corespunzătoare din teoria programării structurate. Mecanismul de execuție este redat în figura 6.2.

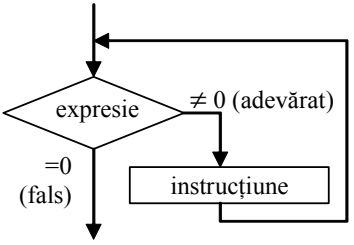


Figura 6.2 - Mecanismul de execuție a instrucțiunii *while*

Exemple:

1.
```
while (a>b)
    a=a+1;
```

2.
```
i=0;
while (i<n)
 {printf("\n %4.2f",v[i]);
  i=i+1;}
```

3.
```
/*prima aparitie a unei valori date*/
#include <stdio.h>
void main()
{ float x,v[10]; unsigned n,i;
  printf("Valoarea lui n:");
  scanf("%i",&n);
  for(i=0;i<n;i++)
   {printf("\nv[%u]=",i+1);
    scanf("%f",&v[i]);}
  printf("dati valoarea:");
  scanf("%f",&x);
  i=0;
  while((i<n)&&(v[i]!=x)) i=i+1;
  if(i>=n) printf("Nu exista valoarea %4.2f",x);
  else printf("Valoarea %4.2f apare prima data pe a %u-a
pozitie",
              x,i+1);}
```

b) *Structura repetitivă condiționată posterior* este implementată (cu unele deosebiri față de teoria programării structurate) prin intermediul instrucțiunii *do-while*. Forma generală este:

```
do instrucțiune
while (expresie);
```

Instrucțiunea *do-while* se execută astfel: se execută *instrucțiune* apoi se evaluează expresia; dacă expresia este nenulă, se repetă procesul, altfel se încheie execuția. Mecanismul de execuție este redat în figura 6.3. Se observă că instrucțiunea *do-while* se abate de la teoria programării structurate, realizînd repetiția pe condiția *adevărată*, în timp ce structura fundamentală o realizează pe condiția *falsă*.

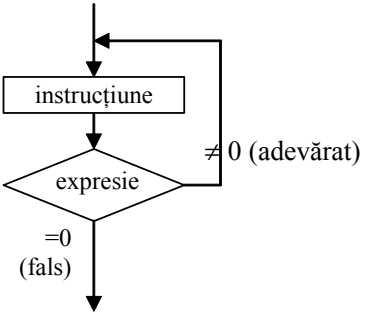


Figura 6.3 - Mecanismul de execuție a instrucțiunii *do-while*

Instrucțiunea *do-while* este echivalentă cu secvența:

```
instrucțiune;
while(expresie)
  instrucțiune;
```

Exemple:

1.
```
do a=a+1;  while (a<=100);
```

2.
```
#include <stdio.h>
main()
{ unsigned i;
  i=1;
  do
   { printf("+");
     printf("-");
     i++;}
  while(i<=80);}
```

c) *Structura repetitivă cu numărător* nu este implementată în C ***(D)***. Ea poate fi simulată prin instrucțiunea *for*, care, datorită facilităților deosebite pe care le oferă, poate fi considerată ca o instrucțiune repetitivă cu totul particulară, nedefinită în teoria programării structurate – ea este mai apropiată de structura *while*.

d) Instrucțiunea *for* din C are următoarea formă generală:

```
for(expresie_1;expresie_2;expresie_3)instrucțiune;
```

Instrucțiunea *for* se execută astfel: se evaluează *expresie_1*; se evaluează *expresie_2* şi, dacă este nulă, se încheie execuția lui *for*, altfel se execută *instrucțiune*, apoi se evaluează *expresie_3*. Se revine la evaluarea lui *expresie_2* ş.a.m.d.

Instrucţiunea *for* realizează structura repetitivă din figura 6.4 şi poate fi înlocuită cu secvenţa:

```
expresie1;
while (expresie2)
 { instrucţiune;
   expresie3;}
```

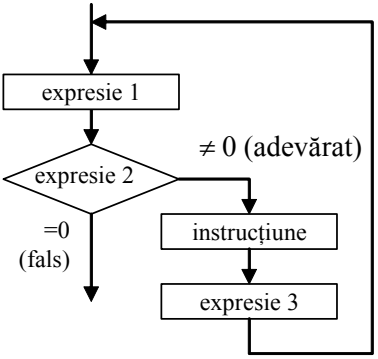


Figura 6.4 - Mecanismul de execuţie a instrucţiunii *for* din C

Pentru a simula structura repetitivă cu numărător, se folosesc forme particulare pentru cele 3 expresii:

- *expresie_1* va fi expresia de iniţializare: *contor=valoare iniţială*;
- *expresie_2* controlează terminarea ciclului: *contor<valoare finală*;
- *expresie_3* realizează avansul contorului: *contor=contor+pas*.

Exemplu:

```
#include <stdio.h>
void main()
{ float i;
  for(i=0;i<10;i=i+1.5)
  printf("\n merge");}
```

În urma execuţiei programului, cuvîntul *merge* se va afişa de şapte ori ([(10-0)/1.5]+1=6+1=7).

Observaţie: *expresie_1*, *expresie_2* şi *expresie_3* pot lipsi, instrucţiunea *for* astfel scrisă definind un ciclu infinit din care se poate ieşi prin alte mijloace decît cele obişnuite.

Exemple:

1.a.

```
/*citirea elementelor unui vector*/
#include <stdio.h>
void main()
{ float v[20];
  int n,i;
  printf("\n Nr. de elemente:");
  scanf("%i",&n);
  for(i=0;i<n;i++)
   { printf("\n v[%i]=",i+1);
     scanf("%f",&v[i]);}
}
```

1.b.

```
/*citirea elementelor unui vector*/
#include <stdio.h>
void main()
{ float v[20];
  int n,i;
  printf("\n Nr. de elemente:");
  scanf("%i",&n);
  for(i=0;i<n;printf("\n v[%i]=",i+1),scanf("%f",&v[i]),i++);
}
```

2.

```
/*sortarea elementelor unui vector*/
#include <stdio.h>
void main()
{ float aux,v[20];
  int n,i;
  unsigned vb;
  printf("\n Nr. de elemente:");
  scanf("%i",&n);
  for(i=0;i<n;printf("\n v[%i]=",i+1),scanf("%f",&v[i]),i++);
  do{vb=0;
     for(i=0;i<n-1;i++)
       if(v[i]>v[i+1])
        {aux=v[i];
         v[i]=v[i+1];
         v[i+1]=aux;
         vb=1;}
    }
  while(vb==1);
  printf("\nVectorul sortat crescator:");
  for(i=0;i<n;printf("%6.2f",v[i]),i++);
  getch();}
```

6.4 Instrucțiuni de salt necondiționat şi ieşire forțată din structuri

Instrucțiunile de salt necondiționat şi ieşire forțată din structuri contravin pricipiilor programării structurate, dar pot fi utilizate în măsura în care, în aplicații complexe, uşurează munca programatorului.

Instrucțiunea ***continue*** se poate folosi numai în interiorul unui ciclu. Prin folosirea ei se abandonează iterația curentă şi se trece la următoarea. Forma ei este:

```
continue;
```

Efectul instrucțiunii este:

- în interiorul instrucțiunilor *while* şi *do-while*: se abandonează iterația curentă şi se trece la evaluarea expresiei care controlează terminarea ciclului;
- în interiorul instrucțiunii for: se abandonează iterația curentă şi se trece la evaluarea lui *expresie_3*.

Instrucținea ***break*** este folosită numai în interiorul unei instrucțiuni structurate şi are ca efect terminarea imediată a execuției instrucțiunii respective.

goto este o instrucţiune de salt necondiţionat moştenită din primele limbaje de programare, care nu foloseau principiile programării structurate. Forma generală este:

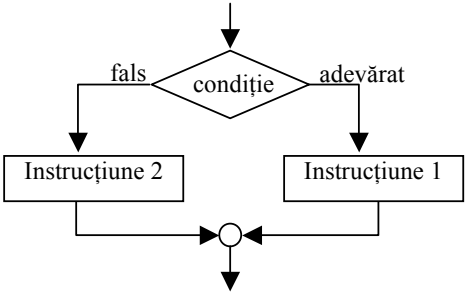
```
goto etichetă;
```

O etichetă este un identificator urmat de caracterul : (două puncte), după care urmează o instrucţiune. Rolul unei etichete este de a defini un punct de salt, către care se poate face trimitere prin instrucţiunea *goto*. Etichetele au valabilitate locală, în corpul funcţiei în care sînt definite. Din acest motiv, nu se poate face salt din corpul unei funcţii la o instrucţiune aflată în altă funcţie.

6.5 Studiul teoretic al structurilor fundamentale

Studiul instrucţiunilor executabile dintr-un limbaj de programare se poate face pornind de la teoria programării structurate. În cele ce urmează se va face o trecere în revistă a structurilor fundamentale şi a modului de implementare a lor în Pascal şi C.

1. Structura alternativă *if-then-else*
   a. Structura fundamentală:



   b. Pascal:

```
if  condiţie then instrucţiune1
                  else instrucţiune2;
```

   c. C:

```
if (expresie) instrucţiune1;
          else instrucţiune2;
```

   d. Concluzii: atît limbajul Pascal cît şi C implementează „natural” structura *if-then-else*; pot fi implementate uşor structurile *if-the* şi *if-else*.

Temă: Vă propunem ca, în mod similar, să continuaţi studiul pentru următoarele structuri: *case-of*, *repetitivă condiţionată anterior*, *repetitivă condiţionată posterior*, *repetitivă cu numărător*. În cadrul studiului evidenţiaţi apropierea implementării structurilor faţă de definirea lor teoretică. Care dintre limbajele Pascal şi C implementează mai „natural” structurile fundamentale? Aveţi o explicaţie?

7. Pointeri

Pointerul este un tip de dată predefinit, care are ca valoare adresa unei zone de memorie.

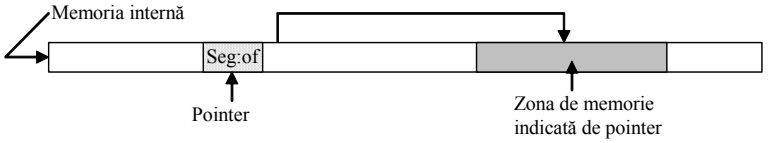


Figura 7.1 - Un pointer este adresa unei alte zone de memorie

Folosirea pointerilor prezintă următoarele avantaje:

- înlocuirea expresiilor cu indici – înmulțirile din formula de calcul a rangului se transformă în adunări şi deplasări;
- posibilitatea alocării dinamice a memoriei;
- folosirea tipurilor procedurale de date;
- calculul adreselor.

În operațiile cu pointeri se folosesc următorii operatori specifici:

Operatori	**C**
Operator de *referențiere*	& (&nume)
Operator de *dereferențiere*	* (*nume sau tip*)

* => extrage conținutul zonei de memorie indicate de pointer;
& => extrage adresa unei variabile (creează o referință).

Cei doi operatori sînt echivalenți cu operatorii din Pascal @ (pentru &) şi, respectiv, ^ (pentru *).

Cei doi operatori au efect invers: *&nume ⇔ nume.

Exemplu: *&*nume* reprezintă: valoarea de la adresa variabilei *nume* (valoarea variabilei *nume*).

7.1 Declararea şi iniţializarea pointerilor

Fie `TIP` un tip de dată oarecare în limbajul C (inclusiv *void*). Declararea `TIP* nume;` este o declaraţie de pointer. `TIP*` este un nou tip de dată denumit *pointer spre* `TIP`, iar *nume* este o variabilă de tipul *pointer spre* `TIP`.

Exemple:

- `int* n;` ⇒ *n* este o variabilă de tip *pointer spre întreg*;
- `struct complex {a,b:real;}* x;` ⇒ *x* este o variabilă de tip *pointer spre o structură de tipul complex*;
- `void* p;` ⇒ *p* este o variabilă de *tip pointer spre void*; *p* poate primi ca valoare adresa unei zone de memorie de orice tip.

Dacă `TIP` este un tip oarecare (mai puţin void) atunci tipul `TIP*` este asemănător tipului *referinţă* din Pascal (este adresa unei zone de memorie de un tip cunoscut; operaţiile care se pot efectua asupra zonei respective de memorie sînt definite de tipul acesteia). Dacă `TIP` este void, atunci `TIP*` este asemănător tipului *pointer* din Pascal (este adresa unei zone de memorie de tip necunoscut; deoarece nu se cunoaşte tipul zonei de memorie, nu sînt definite operaţiile care se pot efectua asupra ei).

Pentru pointerii din exemplele anterioare se rezervă în memoria principală (în segmentul de date) cîte o zonă de 4B în care se va memora o adresă (sub forma *segment:offset*).

Cînd variabila *nume* nu este iniţializată prin declarare, primeşte implicit valoarea *NULL*, similară lui *NIL* din Pascal. La execuţie, poate primi ca valoare adresa unei variabile de tipul *TIP* (numai de tipul *TIP*). Dacă *TIP* este *void*, atunci *nume* poate primi adresa oricărei variabile, de orice tip.

Exemple:

```
int* nume; int a; float b;
nume = &a;    =>este o atribuire corectă; nume are ca valoare adresa variabilei a.
nume = &b;    =>este o atribuire incorectă; nume poate primi ca valoare doar
              adresa unei variabile întregi.
void* nume;
int a; float b;
nume = &a;
nume = &b;    =>ambele atribuiri sînt corecte; nume poate primi ca valoare
              adresa oricărei variabile, de orice tip.
```

Iniţializarea pointerilor se poate realiza ca în exemplul precedent sau, ca şi pentru celelalte variabile, la declarare, astfel:

```
int a; int* nume=&a;
```

Se observă folosirea operatorului de referențiere *&* pentru a crea o referință către variabila *a*. La alocarea dinamică a memoriei se foloseşte o altă metodă pentru inițializarea unui pointer. Operatorul de dereferențiere se utilizează atît pentru definirea tipului pointer, cît şi pentru referirea datelor de la adresa indicată de pointer.

Exemplu:

```
int a,b,c; int* nume;
void* nume2;
b=5;
nume=&a;
*nume=b;
c=*nume+b;
nume2=&b;
*(int*)nume2=10;
c=*(int*)nume2;
```

Se observă folosirea conversiei de tip (*typecasting*), atunci cînd se lucrează cu pointeri spre void (fără tip). Chiar dacă un pointer spre void poate primi ca valoare adresa unei variabile de orice tip, pentru a putea lucra cu ea, este necesară gestionarea corectă a tipului operanzilor.

7.2 Utilizarea pointerilor

7.2.1 Operații cu pointeri

Asupra pointerilor se pot efectua operații aritmetice. Fie secvența:

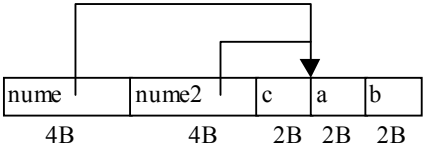
```
int *nume,*nume2, c, a, b; nume=&a; nume2=&a;
```

Incrementare/decrementare

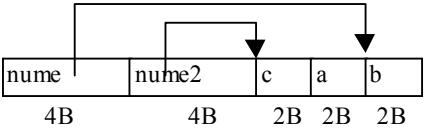
Dacă *nume* este pointer spre un tip *TIP*, prin incrementare/decrementare, valoarea lui *nume* se incrementează/decrementează cu numărul de octeți necesari pentru a memora o dată de tip *TIP*, adică cu `sizeof(TIP)`.

`nume++` ⇔ *nume* are ca valoare o adresă care este incrementată şi primeşte valoarea `nume+sizeof(int)` (care este adresa lui b);
`nume2--` ⇔ *nume* are ca valoare o adresă care este decrementată şi primeşte valoarea `nume-sizeof(int)` (care este adresa lui c);

Situaţia iniţială este următoarea:



După cele 2 operaţii:



Analog se execută operaţiile ++nume şi --nume.

Exemplu:

```
float v[20];
float* p;
int i;
p=&v[i];
```

=> *i* poate avea valori între 0 şi 19

În urma atribuirii *++p* sau *p++*, *p* va avea ca valoare adresa lui v[i] plus 4 octeţi, adică adresa lui v[i+1].

Adunarea/scăderea unui întreg

În general, dacă *p* este un pointer spre un tip *TIP*, atunci cînd se adună un întreg *n* la pointerul *p*, rezultatul va fi tot un pointer spre *TIP*, care are ca valoare adresa memorată în *p*, la care se adună de *n* ori numărul de octeţi necesari pentru a memora o dată de tip *TIP*, adică *n*sizeof(TIP)*. Asemănător se execută scăderea unui întreg dintr-un pointer.

`nume+n` ⇔ *nume* primeşte valoarea `nume+n*sizeof(int)`

`nume-n` ⇔ *nume* primeşte valoarea `nume-n*sizeof(int)`

Exemplu: fie *p* şi *q* pointeri spre tipul *float* (`float* p, *q`). Presupunînd că *p* a fost iniţializat cu valoarea *0x0fff:0x3450*, în urma operaţiei `q=p+3`, *q* primeşte valoarea *0xfff:0x345c* (se adună 3*4 octeţi). În urma operaţiei `q=p-2`, *q* primeşte valoarea *0xffff:0x344a* (se scad 2*4 octeţi).

Operaţiile descrise anterior se folosesc frecvent în lucrul cu masive.

Compararea a doi pointeri

Limbajul C permite compararea a doi pointeri într-o expresie, folosind oricare din operatorii relaţionali (==, !=, <, >, <=, >=).

Rezultatul expresiei *nume op nume2* (unde *op* este unul din operatorii precizați anterior) este *adevărat* (nenul) sau *fals* (zero) după cum *nume* este egal, mai mare sau mai mic decît *nume2*. Doi pointeri sînt egali dacă adresele care constituie valorile lor sînt egale. Privind memoria internă liniar, începînd de la 0x0000:0x0000, un pointer *p* este mai mare decît altul *q*, dacă adresa pe care o conține *p* este mai îndepărtată de începutul memoriei decît adresa conținută de *q*.

Este permisă şi compararea unui pointer cu o valoare constantă. Uzual se foloseşte comparația cu valoarea *NULL* pentru a verifica dacă pointerul a fost inițializat (un pointer neinițializat are valoarea *NULL*), folosind unul din operatorii == sau *!=*. Valoarea *NULL* este definită în *stdio.h* astfel: #define NULL 0

De multe ori se preferă comparația directă cu zero (*nume==0* sau *nume!=0*). În loc de *nume=0* se poate folosi expresia *nume.* Aceasta se interpretează astfel: dacă nume nu a fost inițializat, atunci are valoarea *NULL* (adică 0), deci expresia este falsă. În caz contrar valoarea expresiei este nenulă, deci adevărată. Asemănător se foloseşte expresia *!nume*.

Exemplu:

```
float* p,q,r,t;
float a,b;
p=&a; q=&b; r=&a;
a=5; b=7;
if(t) printf("Pointer initializat!\n");
   else printf("Pointer neinitializat!\n");
if(p==r) printf("Pointeri egali\n");
   else printf("Pointeri diferiti\n");
if(p>q) printf("%d\n",a);
   else printf("%d\n",b);
```

Pe ecran se va afişa:

```
Pointer neinitializat!
Pointeri egali
7
```

deoarece *t* are valoarea *NULL*, variabilele *p şi r* au ca valoare adresa lui *a*, iar *q* conține adresa lui *b*, care este mai mare decît a lui *a* (datorită faptului că *a* a fost alocat primul).

Diferența a doi pointeri

Fie secvența:

```
int m[50],* a, * b;
a=&m[i]; b=&m[j];
```

unde *i* şi *j* sînt întregi în intervalul [0..49]. Expresia *a-b* are valoarea *i-j*, interpretată ca distanța între adresele a şi b, exprimată în zone de memorie de lungime *sizeof(int)*.

Valoarea unei expresii diferenţă se calculează astfel: se face diferenţa între cele două adrese (în octeţi), apoi se împarte la dimensiunea tipului de dată referită de cei doi pointeri (tipul *int* în exemplul de mai sus – vezi figura 7.2). Cei doi pointeri trebuie să refere acelaşi tip de dată, altfel rezultatul nu are semnificaţie. Operaţia este utilă în lucrul cu masive.

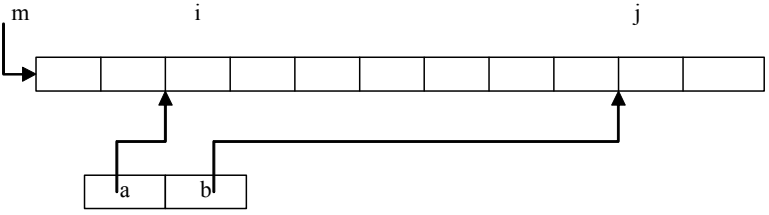


Figura 7.2 - Reprezentarea semnificaţiei variabilelor din exemplul anterior

7.2.2 Legătura între pointeri şi masive

În limbajul C numele unui masiv este un pointer către *tipul de dată al elementelor masivului*.

Pentru masive *unidimensionale*:

```
int m[50];      ⇔ m are tipul int*
int* p;         ⇔ p are tipul int*
```

Diferenţa constă în faptul că zona de memorie către care punctează *m* este rezervată la compilare (ceea ce nu se întîmplă în cazul pointerilor declaraţi ca atare). De aceea *m* nici nu poate primi valori în timpul execuţiei programului (nu se poate schimba adresa memorată în *m*). El memorează adresa primului element din masiv. Referirea unui element *m[i]* este echivalentă cu **(m+i)* – conţinutul de la adresa *m+i*. Limbajul C nu face nici un fel de verificări în privinţa depăşirii limitelor indicilor masivului, de aceea expresiile *m[500]* sau *m[-7]* vor fi considerate corecte de către compilator, existînd riscul unor erori logice. Este sarcina programatorului să se asigure că indicii nu vor depăşi limitele.

Pentru masive *bidimensionale*:

`int m[50][50];` ⇔ *m* are tipul *int*** (pointer la *pointer la int*, cu aceleaşi observaţii ca mai sus) şi semnificaţia următoare: `m[i][j]` ⇔ **(*(i+m)+j)*. Deci, `m[i][j]` reprezintă conţinutul de la adresa *j plus conţinutul de la adresa memorată în i plus m*. Aceasta poate fi interpretată astfel: *m* este un pointer spre un vector de pointeri, fiecare element al vectorului fiind la rîndul lui un pointer spre o linie a matricei (un vector de elemente de tip *float*). În acest fel se alocă matricele în mod dinamic (figura 7.3).

Analog pot fi interpretate masivele cu mai multe dimensiuni.

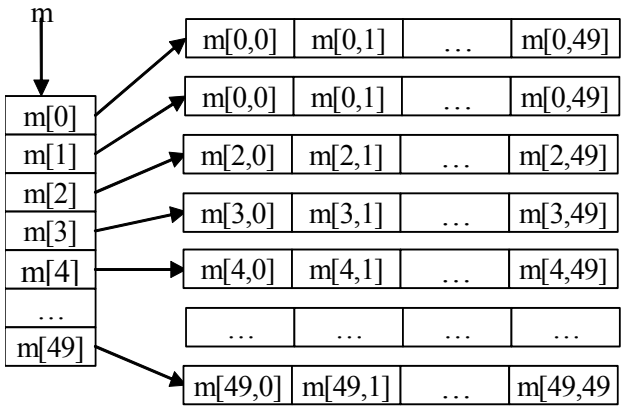


Figura7.3 - Reprezentarea modului de alocare dinamică a spațiului necesar pentru memorarea unei matrice 50x50

Exemple:

- un masiv cu trei dimensiuni *float m[10][10][10]* poate fi interpretat ca un pointer spre un vector de pointeri spre matrice; *m* are tipul *float*** (pointer la pointer la pointer spre tipul *float*).
- un masiv cu *n* dimensiuni este tratat ca un pointer spre un vector de pointeri către masive cu *n-1* dimensiuni.

Pentru a lucra cu elementele unei matrice se poate folosi adresarea indexată (*m[i]* pentru vectori sau *m[i][j]* pentru matrice) sau adresarea elementelor prin pointeri (**(m+i)* pentru vectori sau **(*(m+i)+j)* pentru matrice etc). De asemenea, se poate declara un pointer inițializat cu adresa de început a masivului, iar elementele masivului să fie referite prin intermediul acestui pointer.

Exemple:
```
float* v[10];  float* p;
p=v;
```
După atribuire, pointerul *p* conține adresa de început a masivului şi poate fi folosit pentru referirea elementelor masivului. De exemplu, `v[3]` şi `p[3]` referă aceeaşi zonă de memorie.

Să se scrie secvența de program care citeşte de la tastatură elementele unei matrice, folosind un pointer pentru adresarea elementelor matricei.

```
int m,n;
float a[10][10];
printf("Nr. linii:\n"); scanf("%d", &m);
printf("Nr. coloane:\n"); scanf("%d", &n);
for(i=0;i<m;i++)
    for(j=0;j<n;j++)
        {printf("a(%d,%d)= ",i,j);
          scanf("%f", *(m+i)+j );}  /*  *(m+i)+j  este  un
pointer, care conține adresa lui a[i][j] */
```

Observaţie: în funcţia *scanf* trebuie transmise adrese; în cazul exemplului anterior se putea scrie `&*(*(m+i)+j)`, şi, reducînd, rezultă `*(m+i)+j`.

7.2.3 Simularea transmiterii parametrilor prin adresă

Limbajul C permite transmiterea parametrilor numai prin valoare (la apelul subprogramelor se copiază în stivă valoarea parametrului real şi subprogramul lucrează cu această copie). Subprogramul nu poate modifica valoarea parametrului din apelator ***(D)***.

Dacă parametrul formal este un masiv, el este de fapt un pointer (adresa de început a masivului). Folosind această proprietate, se pot modifica valorile elementelor masivului, iar modificările se vor propaga în blocul apelator, deoarece valoarea care se copiază în stivă este adresa de început a masivului; masivul rămîne în memoria principală şi poate fi modificat prin intermediul adresei sale de început. Astfel se poate simula transmiterea parametrilor prin adresă folosind pointerii. Subprogramul poate modifica valori care să se propage în apelator. În acest scop se transmite ca parametru un pointer spre variabila cu care trebuie să lucreze subprogramul apelat, care va lucra în mod explicit cu pointerul. Un exemplu în acest sens este funcţia de citire a datelor de la tastatură. Parametrii acestei funcţii sînt adresele variabilelor ale căror valori trebuie citite.

Exemplu:
1. Fie un subprogram care calculează suma elementelor unui vector *v* de lungime *n*.

```
void suma(float s, float v[], int n);
{ int i;
  for(s=0,i=0;i<n;i++)
   s+=v[i]; }
```

Subprogramul *suma* calculează suma elementelor vectorului, dar nu poate fi folosit de apelator deoarece valoarea sumei este cunoscută numai în interiorul funcţiei (parametrul a fost transmis prin valoare). În apelator valoarea variabilei corespunzătoare parametrului formal *s* nu va fi modificată. Pentru ca subprogramul să fie utilizabil, trebuie ca parametrul *s* să fie un pointer spre variabila în care se va memora suma elementelor vectorului:

```
void suma(float* s, float v[], int n)
{int i;
  for(s=0,i=0;i<n;i++)  *s+=v[i]; }
```

La apelul funcţiei, primul parametru actual este adresa variabilei în care se memorează suma:

```
void main()
{ float x, m[20];   int n;
//…
  suma(&x, m, n);
//…}
```

2. Să se realizeze un subprogram care citeşte de la tastatură o valoare întreagă care aparţine unui interval dat.

```
void citire(int a, int b, int* x)
{do
     printf("Introduceti numarul: ");
     scanf("%d", x);
 until((*x>=a)&&(*x<=b));}
```

7.2.4 Alocarea dinamică a memoriei

Pentru a memora o valoare de un anumit tip în *heap* este necesar să se declare un pointer către acel tip de dată, apoi să se rezerve memoria necesară ***(A)***. Pentru a rezerva spaţiu în heap se foloseşte funcţia standard:

```
void* malloc(unsigned n);
```

Funcţia rezervă o zonă de *n* octeţi în heap şi returnează adresa acesteia. Deoarece funcţia returnează pointer spre void este necesară conversia spre tipul dorit, astfel:

```
int* nume;
nume=(int *) malloc(sizeof(int));
```

⇔ rezervă în heap spaţiu pentru o valoare de tip întreg.

Eliberarea unei zone de memorie rezervate anterior se face prin funcţia standard:

```
void free(void* p);
```

Funcţia primeşte ca parametru un pointer (indiferent de tip) spre zona de memorie pe care trebuie să o elibereze.

Limbajul C oferă posibilitatea de a aloca contiguu zone de memorie pentru mai multe date de acelaşi tip ***(D)***, prin funcţia standard:

```
void* calloc(unsigned nr_elem, unsigned dim_elem);
```

Funcţia *calloc* rezervă o zonă contiguă de memorie pentru mai multe elemente de acelaşi tip, întorcînd un pointer spre zona respectivă.

```
int* masiv;
masiv=(int*)calloc(50,sizeof(int));
```

⇔ rezervă spaţiu de memorie pentru un vector cu 50 de elemente întregi.

Există şi o variantă a lui *malloc* care returnează în mod explicit un pointer "îndepărtat" (far):

```
void* farmalloc(unsigned long n);
```

Pentru eliberarea unei zone de memorie rezervate prin *farmalloc* se foloseşte funcţia standard:

```
void farfree(void* p);
```

Exemple:
1. Alocarea de spaţiu în heap pentru o matrice.

```
int** m;
int n,p;
/* se alocă spaţiu pentru vectorul cu adresele celor n
linii ale matricei */
m=(int**)malloc(m*sizeof(int*));
for(int i=0;i<m;i++)
   /*se alocă spaţiu pentru fiecare linie a matricei,
   cîte p elemente*/
   m[i]=(int*)malloc(n*sizeof(int));
```

2. Să se scrie un subprogram pentru citirea de la tastatură a dimensiunii şi elementelelor unui vector memorat în heap.

```
void cit_vect(int *n, float *vec)
{ int i;
   printf("Nr. elemente: ");
   scanf("%d ", n);
   v=(float*)malloc(n*sizeof(int));
   for(i=0;i<*n;i++)
      { printf("v(%d)= ",i);
        scanf("%f",&v[i]);}
```

3. Să se scrie o funcţie care să citească cel mult *n* numere întregi şi le păstreze în zona de memorie a cărei adresă de început este dată printr-un pointer. Funcţia returnează numărul valorilor citite.

```
int cit_nr(int n, int* p)
{ int nr, i;
  int* q=p+n;  // q este adresa unde se termina zona
               //rezervata pentru cele n numere
  i=0;
  while(p<q)   //cit timp nu s-au citit n numere
  { printf("Numarul %d= ", i);
    if(scanf("%d", &nr)!=1) break; //in caz de eroare la citire
                                   //se termina ciclul
    *p=nr; p++;    }
  return(i);}
```

7.2.5 Pointeri spre funcții

În limbajul C, numele unei funcții este un pointer care indică adresa de memorie unde începe codul executabil al funcției. Aceasta permite transmiterea funcțiilor ca parametri în subprograme, precum şi lucrul cu tabele de funcții. În acest scop trebuie parcurse următoarele etape:

a. Declararea unei variabile de tip procedural (pointer spre funcție):

```
tip_rezultat (*nume_var)(lista_parametri_formali);
```

unde *nume_var* este o variabilă de tip procedural şi are tipul *pointer spre funcție* cu parametrii *lista_parametri_formali* şi care returnează o valoare de tipul *tip_rezultat.* Lui *nume_var* i se poate atribui ca valoare doar numele unei funcții de prototip:

```
tip_rezultat nume_f(lista_parametrilor_formali);
```

b. Descrierea funcției care utilizează parametrii procedurali:

```
void                                          f(…,tip_rezultat
(*nume)(lista_parametrilor_formali),…)
{ tip_rezultat x;
   …
   x=(*nume)(lista_parametrilor_acuali); …}
```

unde *nume* este parametrul formal de tip procedural.

c. Apelul funcției cu parametri procedurali:

```
tip_rezultat nume_functie(lista_parametrilor_formali)
{  …  }
void main()
{ …
  f(…, nume_functie, …); }
```

Exemplu: Fie o funcție care efectuează o prelucrare asupra unui vector. Nu se cunoaşte apriori tipul prelucrării, aceasta fiind descrisă de o altă funcție, primită ca parametru. Pot exista mai multe funcții care descriu prelucrări diferite asupra unui vector şi oricare din ele poate fi transmisă ca parametru.

```
float suma(float *v, int n)
{ for(int i=0, float s=0; i<n; i++)
     s+=v[i];
  return(s);}
```

```
          float media(float *v, int n)
          { for(int i=0, float m=0; i<n; i++)
               m+=v[i];
            m/=n;
            return(m);}

void functie( float vec[],int dim,float(* prelucrare)(float*, int))
{printf("Rezultatul prelucrarii este: %5.2f\n",(*prelucrare)(vec,dim));
}
```

Apelul se realizează prin transmiterea ca parametru real a funcţiei potrivite prelucrării dorite.

```
void main()
{ float tab[10]; int m,i;
  printf("Numarul de elemente(<10): ");
  scanf("%d ", &m);
  for(i=0,i<m;i++)
    {printf("a(%d)=",i);
     scanf("%f",&tab[i]);}
  printf("Se calculeaza suma elementelor…\n"); functie(tab, m,
suma);
  printf("Se calculeaza media elementelor…\n");
  functie(tab, m, media);
  return;}
```

Lucrul cu pointeri spre funcţii va fi aprofundat în capitolul *Subprograme*.

7.2.6 Detalii despre modificatorul const

În limbajul C constantele simbolice se declară prin directiva de preprocesare *#define*. O altă posibilitate de lucru cu constante este iniţializarea unei variabile cu o valoare şi interzicerea modificării valorii acesteia. În acest scop se foloseşte modificatorul *const*. Sînt permise următoarele forme de utilizare a acestuia:

```
a)          tip const nume = valoare; sau
            const tip nume = valoare;
```

Declaraţia este echivalentă cu `tip nume=valoare`, dar, în plus, nu permite modificarea valorii lui *nume* printr-o expresie de atribuire *nume = valoare_noua*;
Faţă de o constantă simbolică, în acest caz se rezervă spaţiu de memorie în care se înscrie valoarea constantei (constantă obiect).

b) `tip const* nume = valoare;` sau
`const tip* nume = valoare;`

Prin această declarare se defineşte un pointer spre o zonă cu valoare constantă. Nu este permisă atribuirea de genul `*nume=valoare_noua`, dar se poate ca variabilei *nume* să i se atribuie o adresă (de exemplu, *nume* = *p*, unde *p* este un pointer spre *tip*). Pentru a modifica valoarea înscrisă în memorie la adresa memorată de pointerul *nume* se poate folosi totuşi un alt pointer:

```
tip *t;
t=nume;
*t=valoare_noua;
```

c) `const tip* nume;`

Construcția se foloseşte la declararea parametrilor formali, pentru a împiedica modificarea lor în corpul subprogramelor, în cazul în care apelatorul are nevoie de valorile inițiale.

7.2.7 Tratarea parametrilor din linia de comandă

În linia de comandă a unui program pot să apară parametri (sau argumente). Aceştia sînt şiruri de caractere despărțite prin spații. Programul poate accesa aceste argumente prin intermediul parametrilor predefiniți ai funcției *main:*

```
void main(int argc, char* argv[])
```

unde *argc* conține numărul de parametri ai programului, incrementat cu 1.

Exemplu: Dacă programul nu are nici un parametru, *argc* are valoarea 1, dacă programul are doi parametri, *argc* are valoarea 3 etc.

Variabila *argv* este un vector de pointeri care conține adresele de memorie unde s-au stocat şirurile de caractere care constituie parametrii programului. Primul şir (argv[0]) conține identificatorul fişierului (inclusiv calea completă) care memorează programul executabil. Următoarele şiruri conțin parametrii în ordinea în care au apărut în linia de comandă (parametrii în linia de comandă sînt şiruri de caractere separate prin spații). Interpretarea acestor parametri cade în sarcina programului.

Exemplu: Să se scrie un program care afişează parametrii din linia de comandă.

```
#include<stdio.h>
main(int argc, char *argv[]);
{ int i;
  printf("Fisierul executabil: %s\n", argv[0]);
  for(i=1;i<argc;i++)
    printf("Parametrul nr. %d: %s\n",i, argv[i]);}
```

8. Subprograme

8.1 Generalităţi

Conform teoriei programării, subprogramele sînt clasificate în:

- *funcţii*, care returnează un singur rezultat prin "numele" funcţiei şi oricîte prin parametri de ieşire;
- *proceduri*, care returnează oricîte rezultate, prin intermediul parametrilor de ieşire.

Un program C este un ansamblu de *funcţii* care realizează activităţi bine definite. Există o funcţie, numită *main()*, care este apelată la lansarea în execuţie a programului.

Subprogramele C sînt, în mod nativ, funcţii. Pot fi construite subprograme care nu returnează nici un rezultat prin numele lor, comportîndu-se ca o procedură (conform definiţiei din teorie).

Sistemele C au colecţii de "biblioteci" care conţin funcţii standard ***(A)***. Textul sursă al unui program C poate fi partiţionat în mai multe fişiere. Fiecare fişier constă dintr-un set de funcţii şi declaraţii globale. Fişierele care constituie partiţia pot fi compilate şi, eventual, testate separat, dar numai unul va conţine funcţia *main()*.

8.2 Construirea şi apelul subprogramelor

Funcţiile C sînt formate din antet şi un corp. *Antetul* are forma:

```
tip nume([lista-parametri-formali])
```

unde:

- *tip* poate fi un tip simplu de dată. Dacă lipseşte, este considerat tipul implicit (*int* pentru unele compilatoare, v*oid* pentru altele);

- *nume* este un identificator care reprezintă numele funcției;
- *lista-parametrilor-formali* conține parametrii formali sub forma:

```
[tip1 identificator1[,tip2 identificator[,tip3 identificator …]]]
```

Parametrii sînt separați prin virgulă. La limită, lista poate fi vidă. Pentru fiecare parametru trebuie specificat tipul, chiar dacă mai mulți parametri sînt de același tip. Nu este posibilă definirea de liste de parametri cu același tip, ca în Pascal.

Pentru funcțiile care nu întorc o valoare prin numele lor, tipul funcției va fi *void* sau va fi omis.

Corpul este o instrucțiune compusă: conține declarațiile locale și instrucțiunile executabile care implementează algoritmul. Corpul funcției se execută pînă la executarea ultimei instrucțiuni sau pînă la executarea instrucțiunii *return*. Forma ei generală este:

```
return(expresie); sau
return expresie;  sau
return;
```

Prima și a doua formă sînt folosite în cazul funcțiilor care returnează o valoarea prin numele lor. Prin executarea acestei instrucțiuni se evaluează expresia, valoarea sa este atribuită funcției și se încheie execuția funcției. A treia formă este folosită în cazul funcțiilor care nu returnează nici o valoare prin numele lor (poate chiar să lipsească). Dacă este prezentă, efectul ei este încheierea execuției funcției.

Tipul expresiei din instrucțiunea *return* trebuie să coincidă cu tipul funcției.

În limbajul C nu este admisă imbricarea ***(D)*** (definirea unui subprogram în cadrul altui subprogram) și nu sînt permise salturi cu instrucțiunea *goto* în afara subprogramului.

Declararea unui subprogram apare, în cadrul fișierului sursă, înaintea primului apel ***(A)***. Există cazuri particulare în care fie funcțiile se apelează unele pe altele (de exemplu, cazul recursivității mutuale), fie definiția nu se află în fișierul sursă. Pentru a oferi compilatorului posibilitatea să efectueze verificarea validității apelurilor, sînt prevăzute declarații ale subprogramelor fără definire. Aceste declarații se numesc *prototipuri* și apar în afara oricărui corp de funcție. Sintaxa generală este:

```
tip nume ([lista-parametri-formali]);
```

Prototipul este de fapt un antet de funcție după care se scrie caracterul `;` (punct și virgulă). Numele parametrilor pot lipsi, fiind suficientă specificarea tipurilor lor. Folosirea prototipurilor este asemănătoare cu utilizarea clauzei *forward* din *Pascal*. Prototipul trebuie inserat în program înaintea primului apel al funcției. Domeniul de valabilitate a declarației unui subprogram este limitat la partea care urmează declarației din fișierul sursă.

Prototipurile funcțiilor standard se află în fişiere *header* (cu extensia *.h*). Utilizarea unei funcții din bibliotecă impune includerea fişierului asociat, cu directiva *#include*.

Fiind funcții, subprogramele C se apelează ca operanzi în expresii, prin numele funcției urmate de lista parametrilor reali. Expresia care conține apelul poate la limită să conțină un singur operand şi chiar să fie o instrucțiune de tip expresie (vezi capitolul *Operatori şi expresii*). În aceste cazuri valoarea returnată de funcție se pierde, nefiind folosită în nici un fel.

Exemple: Să se scrie o funcție care calculează cel mai mare divizor comun dintre două numere întregi nenule, utilizînd algoritmul lui Euclid şi un apelant pentru testare.

```
#include <stdio.h>
/*definirea functiei cmmdc*/
int cmmdc(int a, int b)
{ int r,d=a,i=b;
  do {r=d%i;
      d=i; i=r;}
  while(r<>0);
  return i;}

void main()
{ int n1,n2;
  printf("Numerele pentru care se va calcula cmmdc:");
  scanf("%d%d",&n1,&n2);
  if(n1&&n2) printf("\ncmmdc=%d",cmmdc(n1,n2));
  else printf("Numerele nu sînt nenule!");}
```

Acelaşi exemplu folosind prototip pentru funcția *cmmdc*:

```
#include <stdio.h>
/* prototipul functiei cmmdc*/
int cmmdc(int, int);

void main()
{ int n1,n2;
  printf("Numerele pentru care se va calcula cmmdc:");
  scanf("%d%d",&n1,&n2);
  if(n1&&n2) printf("\ncmmdc=%d",cmmdc(n1,n2));
  else printf("Numerele nu sînt nenule! ");}

/*definirea functiei cmmdc*/
int cmmdc(int a, int b)
{ int r,d=a,i=b;
  do {r=d%i;
      d=i; i=r;}
  while(r<>0);
return i;}
```

8.3 Transferul datelor între apelant şi apelat

În practica programării, s-au conturat două posibilități de transfer al datelor între apelant şi apelat: prin parametri şi prin variabile globale. Prin utilizarea variabilelor globale nu se face un transfer propriu-zis, ci se folosesc în comun anumite zone de memorie.

8.3.1 Transferul prin parametri

Principial, transferul se poate face prin valoare sau prin adresă. În limbajul C este implementat numai *transferul prin valoare* (valoarea parametrului real este copiată în stivă, iar subprogramul lucrează numai cu această copie). Operațiile efectuate asupra unui parametru formal scalar (care nu este masiv) nu modifică, la ieşirea din subprogram, parametrul real corespunzător.

Transferul valorii este însoțit de eventuale conversii de tip, realizate pe baza informațiilor de care dispune compilatorul despre subprogram. Dacă prototipul precede apelul subprogramului şi nu există o sublistă variabilă de parametri, conversiile se fac similar atribuirilor.

Exemplu:

`tip_returnat nume(tip_parametru p);` ⇔ *p* este transferat prin valoare

Folosind transferul prin valoare se pot transmite numai parametri de intrare în subprogram. Pentru a putea folosi parametri de ieşire trebuie *simulat transferul prin adresă*. În acest scop, se vor efectua explicit operațiile care se fac automat la transferul prin adresă din alte limbaje: se transmite ca parametru adresă parametrului real, iar în subprogram se lucrează cu indirectare.

Exemplu:

`tip_returnat nume(tip_parametru *p);` ⇔ *p* este transferat prin valoare, el este adresa parametrului real.

Pentru parametrii de tip masiv, simularea transferului prin adresă se face în mod implicit, datorită modului de construire a masivelor în C: numele masivului este un pointer. La apel, în stivă se va transfera adresa masivului, iar referirea elementelor se face automat prin calcul de adrese (vezi capitolul *Pointeri)*. Următoarele prototipuri sînt echivalente:

```
tip_returnat nume1(float v[], int n);
tip_returnat nume2(float *v, int n);
```

Exemple:

1. Să se calculeze produsul scalar dintre doi vectori.

a) rezultatul se întoarce prin numele funcției:

```
float ps(float x[], float y[], int n)
{ int i,prod=0;
  for(i=0;i<n;prod+=x[i]*y[i++]);
  return prod;}
```

Apelul se realizează astfel:

```
float a[30],b[30];
int dimensiune;
……
printf("Produsul scalar al vectorilor a si b este:%f",
        ps(a,b,dimensiune));
```

b) rezultatul se întoarce prin parametru de ieşire:

```
void ps(float x[], float y[], int n, float *prod)
{ int i;
  *prod=0;
  for(i=0;i<n;(*prod)+=x[i]*y[i++]);}
```

Apelul se realizează astfel:

```
float a[30],b[30],produs_scalar;
int dimensiune;
ps(a,b,dimensiune,&produs_scalar);
printf("Produsul scalar al vectorilor a si b este:%f",
         produs_scalar);
```

2. Să se calculeze elementul maxim dintr-un vector şi pozițiile tuturor aparițiilor acestuia (*v*, *n* sînt parametri de intrare; *max*, *nr_ap*, *poz* sînt parametri de ieşire).

```
void  maxim(float  v[],int  n,float  *max,int  *nr_ap,int
poz[])
{ int i;
  for(*max=v[0],i=1;i<n;i++)
    if(*max<v[i])
     {*nr_ap=1;poz[0]=i; max=v[i];}
    else if(*max==v[i])poz[*nr_ap++]=i;}
```

Apelul se realizează astfel:

```
float a[30],el_max;
int dimensiune,nr_aparitii,pozitii[30];
maxim(a,dimensiune,&max,&nr_aparitii,pozitii);
```

Antetul subprogramului este echivalent cu construcția

```
void maxim(float *v, int n, float *max, int *nr_ap, int *poz)
```

pentru care corpul subprogramului este acelaşi.

3. Să se calculeze produsul a două matrice.

```
void produs(float a[][10],float b[][20], float c[][20],int m,
            int n,int p)
{ int i,j,k;
  for(i=0;i<m;i++)
    for(j=0;j<p;j++)
      for(c[i][j]=0,k=0;k<n;k++)c[i][j]+=a[i][k]*b[k][j];}
```

Observație: Deşi un tablou nu poate fi returnat ca tip masiv prin numele unei funcții, se pot scrie funcții care returnează prin nume un tablou ca pointer – deorece numele tabloului este echivalent în C cu adresa sa (pointer la începutul masivului). Unui astfel de masiv i se alocă memorie în funcția care îl calculează. Numele său este returnat ca pointer la primul element al tabloului.

Exemple:
1. Să se calculeze produsul dintre o matrice şi un vector.

```
#include<malloc.h>
……….
float * prod(float a[][30], float v[],int m, int n)
{ float *p;int i,j;
  p=(float *)malloc(sizeof(float)*m);
  for(i=0;i<m;i++)
  for(p[i]=0,j=0;j<n;j++) p[i]+=a[i][j]*v[j];
  return p;}
```

Apelul se realizează astfel:

a)

```
float a[20][30], b[30], *c;
int m,n;
……….
c=prod(a,b,m,n);
```

Cu vectorul c se lucrează în modul obişnuit: elementele se referă prin indexare (c[i], i=0..m).

b)

```
float a[20][30], b[30];
int m,n;
……….
```

Se lucrează cu „vectorul” *prod(a,b,m,n)* – elementele sale se referă ca `prod(a,b,m,n)[i]`, i=0..m. Atenție: la fiecare referire de element se apelează şi se execută funcția, ceea ce duce la consum mare şi inutil de resurse. Este preferabilă prima variantă.

2. Să se realizeze un program C pentru ridicarea unei matrice la o putere. Pentru aceasta se folosesc două funcții care returnează, prin pointeri, produsul a două matrice (*înmulțire*), respectiv ridicarea unei matrice la o putere (*putere*).

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>

float** inmultire(float **a,float **b,int n)
{ int i,j,k;  float **c;
  c=(float **)malloc(n*sizeof(float *));
  for(i=0;i<n;i++)
    *(c+i)=(float *)malloc(n*sizeof(float));
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
      for(k=0,c[i][j]=0;k<n;c[i][j]+=a[i][k]*b[k++][j]);
  return c;}

float** putere(float **a,int p,int n)
{ float **c,**ap;int l,m,i;
  ap=(float **)malloc(n*sizeof(float *));
  for(i=0;i<n;i++)
  *(ap+i)=(float *)malloc(n*sizeof(float));
  for(l=0;l<n;l++)
    for(m=0;m<n;ap[l][m]=a[l][m],m++);
  for(i=0;i<p-1;i++)
    {c=inmultire(a,ap,n);
     for(l=0;l<n;l++)
       for(m=0;m<n;ap[l][m]=c[l][m],m++);}
  return ap;}

void main()
{ int i,j,p,n,l,m;  float **a,**ap,f;
  clrscr();
  printf("\n n=");
  scanf("%i",&n);
  a=(float **)malloc(n*sizeof(float *));
  for(i=0;i<n;i++)
    *(a+i)=(float *)malloc(n*sizeof(float));
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
      {scanf("%f ",&f);
       *(*(a+i)+j)=f;}
  scanf("%i",&p);
  ap=putere(a,p,n);
  for(i=0;i<n;i++)
    {for(j=0;j<n;j++)
     printf("%f ",*((*(ap+i)+j)));
     printf("\n");}
  getch();}
```

8.3.2 Transferul prin variabile globale

Variabilele *globale* se declară în afara funcțiilor, inclusiv în afara rădăcinii. Ele pot fi referite din orice alte funcții, inclusiv din rădăcină. De aceea, schimbul de valori între apelant şi apelat se poate realiza prin intermediul lor. Variabilele declarate într-o funcție se numesc *locale* (din clasa *automatic*) şi pot fi referite numai din funcția respectivă. Domeniul de valabilitate a unei variabile locale este *blocul* (funcția sau instrucțiunea compusă) în care a fost definită.

Exemplu:

```
#include <stdio.h>
int a;
float z(char b)
{ int b;
  ……}
main()
{ int c;
  ……
  { /* instructiunea compusa r */
    int d;
    ……}
}
```

Domeniile de valabilitate a referirilor variabilelor declarate sînt: *b* poate fi referit doar în funcția *z*; *c* poate fi referit doar în funcția *main*; *d* poate fi referit doar în instrucțiunea compusă *r*; *a* este globală şi poate fi referită de oriunde.

8.4 Pointeri spre funcții

Limbajul C permite lucrul cu variabile de tip pointer, care conțin adresa de început a unei funcții (a codului său executabil). Aceste variabile permit transferul adresei funcției asociate ca parametru, precum şi apelul funcției prin intermediul pointer-ului său.

Următoarea declarație defineşte *pointer_f* ca pointer spre funcția cu rezultatul *tip_returnat* şi parametrii *parametri*.

```
tip_returnat (*pointer_f)([parametri])
```

Observație*:* Nu trebuie să se confunde un pointer la o funcție cu o funcție care are ca rezultat un pointer, cu sintaxa de forma `tip_returnat *pointer_f([parametri])`.

Adresa unei funcții se obține prin simpla specificare a identificatorului acesteia (fără specificarea parametrilor sau parantezelor) şi poate fi atribuită unui pointer spre funcție cu rezultat şi parametri compatibili. Pointerul poate fi folosit ulterior pentru apelul funcției sau transmis ca parametru real în apelul unui subprogram care conține, în lista parametrilor formali, un pointer la un prototip de funcție compatibilă.

Exemple:

1. Să se aproximeze soluția unei ecuații de forma f(x)=0 prin metoda bisecției.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
/*prototipul functiei bisectie*/
void   bisectie(float,float,float(*f)(float),float,long,int   *,float
*);

/*prototipul functiei pentru care se aplica metoda bisectiei*/
float fct(float);

/* functia principala*/
void main()
{ float a,b,eps,x;
  int cod;long n;
  float (*functie)(float);
  clrscr();
  printf("Introduceti capetele intervalului:");
  scanf("%f%f",&a,&b);
  printf("\nEroarea admisa:");
  scanf("%f",&eps);
  printf("\nNumarul maxim de termeni construiti:");
  scanf("%li",&n);
  functie=fct;
  bisectie(a,b,functie,eps,n,&cod,&x);
  if(!cod)printf("\nNu se poate calcula solutia aproximativa");
  else printf("\n Solutia aproximativa este: %f",x);}

/*descrierea functiei pentru care se aplica metoda bisectiei*/
float fct(float x)
{ return x*x*x-3*x+14;}

/*functia ce implementeaza metoda bisectiei*/
void bisectie(float a,float b,float (*f)(float),float eps,long n,
              int *cod,float *x)
{ int gata=0;
  long c;
  for(c=0;(c<n)&&!gata;c++)
    {*x=(a+b)/2;
     gata=fabs(*x-a)<eps;
     if ((*f)(*x)*(*f)(a)<0)b=*x;
     else a=*x;}
  *cod=gata;}
```

2. Să se sorteze un şir cu elemente de un tip neprecizat, dar pe care se poate defini o relaţie de ordine (de exemplu numeric, şir de caractere, caracter).
Metoda aleasă spre exemplificare este sortarea prin selecţie directă. Un subprogram de sortare care să nu depindă de tipul elementelor şi de criteriul de sortare considerat trebuie să aibă ca parametri formali:
- vectorul de sortat, ca pointer la tipul *void*, asigurîndu-se astfel posibilitatea realizării operaţiei de schimbare a tipului ("cast") în funcţie de necesităţile ulterioare (la momentul apelului se poate realiza modificarea tipului `*void` în `*tip_element`, unde *tip_element* reprezintă tipul elementelor vectorului de sortat);
- dimensiunea vectorului de sortat şi numărul de octeţi din reprezentarea tipului elementelor vectorului;
- pointerul la o funcţie de comparare, cu argumente de tip `*void`, care să permită la apel atît schimbarea de tip, cît şi descrierea efectivă a relaţiei de ordine.

Cum tipul elementelor vectorului nu este cunoscut la momentul descrierii procedurii de sortare, operaţia de atribuire nu poate fi folosită, ea fiind înlocuită de o funcţie de copiere a unui număr prestabilit de octeţi, de la o adresă sursă la una destinaţie. O astfel de funcţie există în biblioteca *mem.h*, sintaxa ei fiind:

```
void *memmove(void *destinaţie, const void *sursă, unsigned
n)
```

Pentru accesarea elementului de rang *i* din vector se foloseşte formula *v+i*nr_octeti*. Fişierul sursă care conţine funcţia de sortare descrisă anterior este:

```
//fisier exp_tip.cpp
#include <mem.h>
include<malloc.h>
int compara(const void *x, const void *y);
void sort(void *v, int n, int dim,
          int (*compara)(const void *x,const void *y))
{ int i,j;
  void *aux;
  aux=malloc(dim);
  for(i=0;i<n-1;i++)
    for(j=i+1;j<n;j++)
      if((*compara)((char*)v+dim*i,(char*)v+dim*j))
       {memmove(aux,(char*)v+dim*i,dim);
        memmove((char*)v+dim*i,(char*)v+dim*j,dim);
        memmove((char*)v+dim*j,aux,dim);}
 free(aux);}
```

Exemplu de apel pentru un vector de numere reale:

```
#include <stdio.h>
#include<conio.h>
#include "exp_tip.cpp"
```

```
int compara(const void *a, const void *b)
{ if(*(float *)a>*(float *)b)return 1;
  else return 0;}
void main()
{ float vect[20]; int n,i;
  clrscr();
  printf("Dimensiunea vectorului:");scanf("%d",&n);
  printf("\nElementele:");
  for(i=0;i<n;i++) scanf("%f",&vect[i]);
  sort(vect,n,sizeof(float),compara);
  printf("\nElementele sortate:");
  for(i=0;i<n;i++) printf("\n%f",vect[i]);
  getch();}
```

Exemplu de apel pentru un vector de cuvinte:

```
#include <stdio.h>
#include <string.h>
#include<conio.h>
#include "exp_tip.cpp"
int compara(const void *a, const void *b)
{ if(strcmp((char *)a, (char *)b)>0)return 1;
  else return 0;}

void main()
{ typedef char cuvant[10];
  cuvant vect[20];
  int n;
  clrscr();
  printf("Dimensiunea vectorului de cuvinte:");
  scanf("%d",&n);
  printf("\nCuvintele:");
  for(int i=0;i<n;i++)scanf("%s",&vect[i]);
  sort(vect,n,10,compara);
  printf("\nCuvintele sortate:");
  for(i=0;i<n;i++)printf("\n%s",vect[i]);
  getch();}
```

8.5 Funcții cu număr variabil de parametri

Bibliotecile limbajului C conțin subprograme standard cu număr variabil de parametri ***(A)***. Spre deosebire de limbajul Pascal, limbajul C permite definirea funcțiilor utilizator cu număr variabil de parametri, prin utilizarea unui set de macrodefiniții, declarate în biblioteca *stdarg.h*, care permit accesul la lista de parametri.

Fişierul *stdarg.h* declară tipul *va_list* şi funcțiile *va_start*, *va_arg* şi *va_end*, în care:

- *va_list* este un pointer către lista de parametri. În funcția utilizator corespunzătoare trebuie declarată o variabilă (numită în continuare *ptlist*) de acest tip, care va permite adresarea parametrilor;

- *va_start* inițializează variabila *ptlist* cu adresa primului parametru din sublista variabilă. Prototipul acestei funcții este:

```
void va_start(va_list ptlist, ultim)
```

unde *ultim* reprezintă numele ultimului parametru din sublista variabilă. În unele situații (vezi exemplele), se transferă în acest parametru numărul de variabile trimise.

- *va_arg* întoarce valoarea parametrului următor din sub-lista variabilă. Prototipul acestei funcții este:

```
tip_element va_arg(va_list ptlist, tip_element)
```

unde *tip_element* este tipul elementului transferat din listă. După fiecare apel al funcției *va_arg*, variabila *ptlist* este modificată astfel încît să indice următorul parametru.

- *va_end* încheie operația de extragere a valorilor parametrilor şi trebuie apelată înainte de revenirea din funcție. Prototipul funcției este:

```
void va_end(va_list ptlist)
```

Problema numărului de parametri şi tipurilor lor este tratată de programator.

Exemple:

1. Să se calculeze cel mai mare divizor comun al unui număr oarecare de numere întregi.

```
#include<stdio.h>
#include<conio.h>
#include<stdarg.h>

int cmmdc_var(int,...);
int cmmdc(int, int);

void main()
{ int x,y,z,w;
  clrscr();
  scanf("%d%d%d%d",&x,&y,&z,&w);
  printf("\nCmmdc al primelor 3 numere:%d\n",cmmdc_var(3,x,y,z));
  printf("\nCmmdc al tuturor numerelor:%d\n",cmmdc_var(4,x,y,z,w));}

//cel mai mare divizor comun a doua numere
int cmmdc(int x,int y)
{ int d=x,i=y,r;
  do{r=d%i;
     d=i;i=r;}
  while(r);
  return d;}
```

```
//cel mai mare divizor comun a nr numere
int cmmdc_var(int nr,...)
{ va_list ptlist;
  /*initializarea lui ptlist cu adresa de  inceput a listei de
parametri*/
  va_start(ptlist,nr);
  //extragerea primului parametru, de tip int
  x=va_arg(ptlist,int);
  for(int i=1;i<nr;i++){
  //extragerea urmatorului element din lista de parametri
  y=va_arg(ptlist,int);
  z=cmmdc(x,y);x=z;}
  va_end(ptlist);
  return x;}
```

2. Să se interclaseze un număr oarecare de vectori.

Spre deosebire de exemplul anterior, în care în lista de parametri a funcției cu număr oarecare de parametri figurau elemente de acelaşi tip (*int*), acest exemplu ilustrează modul de transfer şi acces la elemente de tipuri diferite. Funcției *intre_var* i se transmit la apel vectorul rezultat, iar pentru fiecare vector de interclasat, adresa de început (pointer la tipul *double*) şi numărul de elemente (int). Numărul parametrilor din lista variabilă este, în acest, caz 2*numărul de vectori de interclasat.

```
#include<stdarg.h>
#include<stdio.h>
#include<conio.h>

void inter(double *,int,double *,int,double *);
void inter_var(double *,int nr,...);

void main()
{ int n1,n2,n3,n4; double x1[10],x2[10],x3[10],x4[10],z[50];
  clrscr();
  scanf("%d%d%d%d",&n1,&n2,&n3,&n4);
  for(int i=0;i<n1;i++)scanf("%lf",&x1[i]);
  for(i=0;i<n2;i++)scanf("%lf",&x2[i]);
  for(i=0;i<n3;i++)scanf("%lf",&x3[i]);
  for(i=0;i<n4;i++)scanf("%lf",&x4[i]);
  inter_var(z,4,x1,n1,x2,n2);
  printf("\nRezultatul interclasarii primilor 2 vectori\n");
  for(i=0;i<n1+n2;i++)
    printf("%lf ",z[i]);
  inter_var(z,8,x1,n1,x2,n2,x3,n3,x4,n4);
  printf("\nRezultatul interclasarii celor 4 vectori\n");
  for(i=0;i<n1+n2+n3+n4;i++)
    printf("%lf ",z[i]);}
void inter(double *x, int n1, double *y, int n2, double *z)
{ int i,j,k;
  for(i=0,j=0,k=0;(i<n1)&&(j<n2);k++)
    if(x[i]<y[j])z[k]=x[i++];
    else z[k]=y[j++];
  if(i<n1)for(;i<n1;z[k++]=x[i++]);
  else for(;j<n2;z[k++]=y[j++]);}
```

```
void inter_var(double *z,int nr,...)
{ va_list ptlist;
  double *x,*y,x1[100];
  int n1,n2;
  /*initializarea lui ptlist cu adresa de inceput a listei de
parametri*/
  va_start(ptlist,nr);
  //extragerea primului vector
  x=va_arg(ptlist,double *);
  //extragerea dimensiunii lui
  n1=va_arg(ptlist,int);
  for(int j=0;j<n1;j++)x1[j]=x[j];
  for(int i=1;i<(int)(nr/2);i++)
    {//extragerea urmatorului vector
     y=va_arg(ptlist,double *);
     //extragerea numarului sau de elemente
     n2=va_arg(ptlist,int);
     inter(x1,n1,y,n2,z);
     for(j=0;j<n1+n2;j++)x1[j]=z[j];n1+=n2;}
  va_end(ptlist);}
```

9. Fişiere de date

9.1 Elemente generale

Indiferent de limbajul de programare folosit, operaţiile necesare pentru prelucrarea fişierelor sînt:

- descrierea fişierului (crearea tabelei care memorează caracteristicile fişierului);
- asignarea fişierului intern (numele logic) la unul extern (fizic);
- deschiderea fişierului;
- operaţii de acces la date („articole”);
- închiderea fişierului.

Pentru lucrul cu fişiere trebuie identificate tipurile acestora, metodele de organizare, modurile de acces şi tipurile de articole acceptate. Din punct de vedere al tipurilor de date, în C există un singur tip de fişiere ***(D)***: flux de octeţi (înşiruire de octeţi, fără nici un fel de organizare sau semnificaţie). Organizarea acestui flux de octeţi este secvenţială ***(A)***. Accesul la fişiere se poate face secvenţial sau direct (cu excepţia fişierelor standard, la care accesul este numai secvenţial). În bibliotecile limbajului există funcţii predefinite pentru prelucrarea fişierelor. Funcţiile de prelucrare la nivel superior a fişierelor tratează fluxul de octeţi acordîndu-i o semnificaţie oarecare. Putem spune că, din punctul de vedere al prelucrării, la acest nivel ne putem referi la fişiere text şi fişiere binare ***(A)***.

Există fişiere standard, care sînt gestionate automat de sistem, dar asupra cărora se poate interveni şi în mod explicit ***(A)***. Acestea sînt:

- fişierul standard de intrare (*stdin*);
- fişierul standard de ieşire (*stdout*);
- fişierul standard pentru scrierea mesajelor de eroare (*stderr*);
- fişierul standard asociat portului serial (*stdaux*);
- fişierul standard asociat imprimantei cuplate la portul paralel (*stdprn*).

Fişierele standard pot fi redirectate conform convenţiilor sistemului de operare, cu excepţia lui *stderr* care va fi asociat întotdeauna monitorului.

În lucrul cu fişiere (sau la orice apel de sistem), în caz de eroare în timpul unei operaţii se setează variabila *errno*, definită în *errno.h*, *stddef.h* şi *stdlib.h*. Valorile posibile sînt definite în *stdlib.h*.

9.2 Operaţii de prelucrare a fişierelor

În limbajul C există două niveluri de abordare a lucrului cu fişiere: *nivelul inferior* de prelucrare (fără gestiunea automată a zonelor tampon de intrare/ieşire) şi *nivelul superior* de prelucrare (se folosesc funcţii specializate de gestiune a fişierelor). În continuare, prin specificator de fişier se va înţelege un nume extern de fişier, conform convenţiilor sistemului de operare. Specificatorul de fişier poate să conţină strict numele fişierului sau poate conţine şi calea completă pînă la el.

9.2.1 Nivelul inferior de prelucrare a fişierelor

Nivelul inferior de prelucrare este folosit rar, numai în programele de sistem.

La acest nivel, descrierea fişierelor se realizează în corpul programelor, caracteristicile acestora obţinîndu-se din context. Maniera de prelucrare este asemănătoare celei de la nivelul sistemului de operare. Nu există un tip anume de dată, fişierul fiind referit printr-un index care indică intrarea într-o tabelă de gestiune a resurselor sistemului de operare. Acest index este de tip *int* şi se numeşte manipulator de fişier (*handle*). Manipulatorul este creat şi gestionat de către sistemul de operare. Utilizatorul îl foloseşte pentru a indica sistemului fişierul asupra căruia doreşte să facă prelucrări.

Pentru utilizarea acestui nivel, în programul C trebuie incluse bibliotecile standard *io.h*, *stat.h* şi *fcntl.h*.

Crearea şi *asignarea* unui fişier nou se realizează prin apelul funcţiei *creat*, care are următorul prototip:

```
int creat(const char* numef, int protecţie);
```

Funcţia returnează manipulatorul fişierului nou creat; *numef* este un pointer spre un şir de caractere care defineşte specificatorul de fişier, iar *protecţie* defineşte modul de protecţie a fişierului creat (protecţia este dependentă de sistemul de operare). În biblioteca *stat.h* sînt definite următoarele valori pentru parametrul protecţie: S_IREAD (citire), S_IWRITE (scriere), S_IEXEC (execuţie). Aceste valori pot fi combinate folosind operatorul | (sau logic pe biţi). Funcţia *creat* poate fi apelată şi pentru un fişier existent, efectul fiind acelaşi cu apelul procedurii *rewrite*

din Pascal (se şterge fişierul existent şi se creează unul gol, cu acelaşi nume; conţinutul fişierului existent se pierde). În caz de eroare se returnează valoarea –1 şi se setează variabila globală *errno*, care defineşte tipul erorii. Valorile obişnuite pentru *errno* sînt EBADF (manipulator eronat, nu a fost găsit fişierul) sau EACCES (fişierul nu poate fi accesat).

Deschiderea unui fişier existent se realizează prin apelul funcţiei *open*, care are următorul prototip:

```
int open(const char *path,int access[,unsigned mod]);
```

Funcţia returnează manipulatorul fişierului; *numef* este pointer spre un şir de caractere care defineşte specificatorul de fişier; *acces* este modul de acces la fişier; constantele care descriu modurile de acces la fişier sînt descrise în *fcntl.h*. Cele mai importante sînt: O_RDONLY – fişierul va fi accesat numai pentru citire; O_WRONLY – fişierul va fi accesat numai pentru scriere; O_RDWR – fişierul va fi accesat atît pentru citire cît şi pentru scriere; O_CREAT: fişierul va fi creat ca nou. Aceste moduri pot fi combinate folosind operatorul |. *Mod* este folosit numai dacă parametrul *acces* conţine şi valoarea O_CREAT, caz în care indică modul de protecţie a acestuia: S_IWRITE – se permite scrierea în fişier; S_IREAD – se permite citirea din fişier; S_IREAD|S_IWRITE – se permite atît scrierea, cît şi citirea din fişier.

Citirea dintr-un fişier se realizează prin apelul funcţiei *read*, care are următorul antet:

```
int read(int nf, void* zonat, unsigned n);
```

Funcţia returnează numărul de octeţi citiţi din fişier; *nf* este manipulatorul de fişier (alocat la crearea sau deschiderea fişierului), *zonat* este un pointer spre zona tampon în care se face citirea (aceasta este definită de programator), iar *n* este dimensiunea zonei receptoare (numărul maxim de octeţi care se citesc). Numărul maxim de octeţi care pot fi citiţi este 65534 (deoarece 65535 – 0xFFF – se reprezintă intern la fel ca -1, indicatorul de eroare). În cazul citirii sfîrşitului de fişier se va returna valoarea 0 (0 octeţi citiţi), iar la eroare se returnează -1 (tipul erorii depinde de sistemul de operare). Fişierul standard de intrare (*stdin*) are descriptorul de fişier 0.

Scrierea într-un fişier se realizează prin apelul funcţiei *write*, care are următorul prototip:

```
int write(int nf, void* zonat, unsigned n);
```

Funcţia returnează numărul de octeţi scrişi în fişier; *nf* este manipulatorul de fişier (alocat la crearea sau deschiderea fişierului), *zonat* este un pointer spre zona tampon din care se face scrierea (aceasta este definită de programator); *n* este numărul de octeţi care se scriu. Numărul maxim de octeţi care pot fi citiţi

este 65534 (deoarece 65535 – 0xFFF – se reprezintă intern la fel ca -1, indicatorul de eroare). În general, trebuie ca la revenirea din funcția *write*, valoarea returnată să fie egală cu *n*; dacă este mai mică, s-a produs o eroare (probabil discul este plin). La scrierea în fişiere text, dacă în fluxul octeților care se scriu apare caracterul LF, *write* va scrie în fişier perechea CR/LF. În caz de eroare, valoarea returnată este -1 şi se setează variabila *errno*. Fişierul standard de ieşire (*stdout*) are manipulatorul 1, iar cel de eroare (*stderr*) are manipulatorul 2.

Închiderea unui fişier se realizează prin apelul funcției *close*, care are următorul prototip:

```
int close(int nf);
```

Funcția returnează valoarea 0 (închidere cu succes) sau -1 (eroare); *nf* este manipulatorul de fişier. De asemenea, închiderea unui fişier se realizează automat, dacă programul se termină prin apelul funcției *exit*.

Poziționarea într-un fişier se realizează prin apelul funcției *lseek*, care are următorul prototip:

```
long lseek(int nf, long offset, int start);
```

Funcția returnează poziția față de începutul fişierului, în număr de octeți; *nf* este manipulatorul de fişier; *offset* este un parametru de tip long (numărul de octeți peste care se va deplasa pointerul în fişier), iar *start* este poziția față de care se face deplasarea: 0 (începutul fişierului), 1 (poziția curentă în fişier) sau 2 (sfîrşitul fişierului). La eroare returnează valoarea -1L.

Exemple:
1. Apelul `vb=lseek(nf, 0l, 2);` ⇔ realizează poziționarea la sfîrşitul fişierului (în continuare se poate scrie în fişier folosind write);
2. Apelul `vb=lseek(nf, 0l, 0);` ⇔ realizează poziționarea la începutul fişierului.

Ştergerea unui fişier existent se realizează prin apelul funcției *unlink*, care are următorul prototip:

```
int unlink(const char* numef);
```

Funcția returnează 0 (ştergere cu succes) sau -1 (eroare); *numef* este un pointer spre un şir de caractere care defineşte specificatorul de fişier. În caz de eroare se setează variabila *errno* cu valoarea ENOENT (fişierul nu a fost găsit) sau EACCES (accesul interzis pentru această operație, de exemplu pentru fişiere *read only*). Pentru a putea şterge un fişier *read only* trebuie întîi schimbate drepturile de acces la fişier, folosind funcția *chmod*:

```
int chmod(const char *cale, int mod);
```

unde *cale* este specificatorul de fişier, iar *mod* noile permisiuni. Permisiunile sînt aceleaşi ca la funcţia *open*. Rezultatul întors de *chmod* are aceeaşi semnificaţie ca şi *unlink*.

Verificarea atingerii sfîrşitului de fişier se face folosind funcţia *eof*:

```
int eof(int nf);
```

unde *nf* este manipulatorul fişierului. Funcţia returnează valoarea 1 dacă pointerul este poziţionat pe sfîrşitul fişierului, 0 în caz contrat şi -1 în caz de eroare (nu este găsit fişierul – *errno* primeşte valoarea EBADF).

Exemplu:

```
#include <sys\stat.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{ int handle;
  char msg[] = "This is a test";
  char ch;
  /* create a file */
  handle = open("TEST.$$$", O_CREAT | O_RDWR, S_IREAD | S_IWRITE);
  /* write some data to the file */
  write(handle, msg, strlen(msg));
  /* seek to the begining of the file */
  lseek(handle, 0L, SEEK_SET);
  /* reads chars from the file until we hit EOF */
  do {read(handle, &ch, 1);
      printf("%c", ch);}
  while (!eof(handle));
  close(handle);
  return 0;}
```

Bibliotecile limbajului conţin şi alte funcţii pentru prelucrarea fişierelor la nivel inferior, inclusiv variante ale funcţiilor anterioare, apărute o dată cu dezvoltarea sistemelor de operare.

9.2.2 Nivelul superior de prelucrare a fişierelor

La acest nivel, un fişier se descrie ca *pointer* către o structură predefinită (*FILE* – tabela de descriere a fişierului (FIB)):

```
FILE* f;
```

Tipul FILE (descris în *stdio.h*) depinde de sistemul de operare.

Fişierul este considerat ca flux de octeți, din care funcțiile de prelucrare preiau secvențe pe care le tratează într-un anumit fel (sau în care inserează secvențe de octeți).

Funcțiile folosite la acest nivel pot fi împărțite în trei categorii: funcții de prelucrare generale, funcții de citire/scriere cu conversie şi funcții de citire/scriere fără conversie. Funcțiile de prelucrare generală se aplică tuturor fişierelor, indiferent de tipul informației conținute; prelucrarea efectuată de acestea nu are nici un efect asupra conținutului fişierului. Funcțiile care lucrează cu conversie se aplică fişierelor care conțin informație de tip text (linii de text, separate prin perechea CR/LF, iar la sfîrşit se găseşte caracterul CTRL-Z). Funcțiile care lucrează fără conversie se aplică fişierelor care conțin informație binară.

Funcțiile de citire/scriere deplasează pointerul de citire/scriere al fişierului, spre sfîrşitul acestuia, cu un număr de octeți egal cu numărul de octeți transferați (fără a trece de sfîrşitul de fişier).

Funcții de prelucrare generală

Deschiderea şi *asignarea* se realizează prin apelul funcției *fopen*. Funcția returnează un pointer spre o structură de tip FILE (în care sînt înscrise date referitoare la fişierul deschis) sau NULL dacă fişierul nu se poate deschide:

```
FILE* fopen(const char* nume_extern,const char* mod);
```

Parametrul *nume_extern* constituie specificatorul de fişier iar *mod* este un şir de caractere care specifică modul de deschidere a fişierului. Asignarea se realizează prin expresie de atribuire de tipul:

```
nume_intern=fopen(sir_nume_extern,sir_mod);
```

Exemplu:
```
FILE* f;
f = fopen("PROD.DAT","r");
```

Modurile în care poate fi deschis un fişier sînt prezentate în tabelul 9.1.

Tabelul 9.1 Modurile de deschidere a unui fişier

Mod	Scop
a	Deschide un fişier existent pentru adăugare la sfîrşit (extindere) sau îl creează dacă nu există. Este permisă numai scrierea. Numai pentru fişiere text.
r	Deschide un fişier existent numai pentru citire
w	Suprascrie un fişier existent sau creează unul nou, permițîndu-se numai operația de scriere
a+	Deschide un fişier existent pentru adăugare la sfîrşit (extindere) sau îl creează dacă nu există. Sînt permise citiri şi scrieri. Numai pentru fişiere text.
r+	Deschide un fişier existent pentru citire şi scriere
w+	Suprascrie un fişier existent sau creează unul nou, permițîndu-se atît citiri, cît şi scrieri

La opţiunile de mai sus se poate adăuga *b* pentru fişiere binare sau *t* pentru fişiere text. Dacă nu este prezentă nici litera *b* nici litera *t*, modul considerat depinde de valoarea variabilei *_fmode*: dacă valoarea este O_BINARY, se consideră fişier binar; dacă valoarea este O_TEXT, se consideră fişier text. De obicei implicită este valoarea O_TEXT.

Modurile uzuale pentru deschiderea fişierelor sînt prezentate în tabelul 9.2.

Tabelul 9.2 Moduri uzuale pentru deschiderea fişierelor

Operaţia de gestiune	Fişiere text	Fişiere binare
Creare	w	wb
Consultare	r	rb
Actualizare	nu	r+b
Creare şi actualizare	w+	rwb, w+b
Extindere	a	nu

Închiderea fişierelor se realizează prin apelul funcţiei *fclose*, care are următorul prototip:

```
int fclose(FILE* f);
```

Funcţia închide fişierul primit ca parametru şi returnează valoarea 0, în caz de succes, sau -1, în caz de eroare. Înainte de închiderea fişierului, sînt golite toate bufferele asociate lui. Bufferele alocate automat de sistem sînt eliberate.

Revenirea la începutul fişierului se realizează prin funcţia *rewind*, cu prototipul:

```
void rewind(FILE *f);
```

Executarea funcţiei are ca efect poziţionarea la începutul fişierului *f* (care era deschis anterior), resetarea indicatorului de sfîrşit de fişier şi a indicatorilor de eroare (se înscrie valoarea 0). După apelul lui *rewind* poate urma o operaţie de scriere sau citire din fişier.

Testarea sfîrşitului de fişier se realizează prin apelul macrodefiniţiei *feof*:

```
int feof(FILE* f);
```

Macro-ul furnizează valoarea indicatorului de sfîrşit de fişier asociat lui *f*. Valoarea acestui indicator este setată la fiecare operaţie de citire din fişierul respectiv ***(D)***. Valoarea întoarsă este 0 (*fals*) dacă indicatorul are valoarea *sfîrşit de fişier* şi diferit de zero (*adevărat*) în caz contrar. Apelul lui *feof* trebuie să fie precedat de apelul unei funcţii de citire din fişier. După atingerea sfîrşitului de fişier, toate încercările de citire vor eşua, pînă la apelul funcţiei *rewind* sau închiderea şi redeschiderea fişierului.

Golirea explicită a zonei tampon a unui fişier se realizează prin apelul funcției *fflush*, care are următorul prototip:

```
int fflush(FILE* f);
```

Dacă fişierul *f* are asociat un buffer de ieşire, funcția scrie în fişier toate informațiile din acesta, la poziția curentă. Dacă fişierul are asociat un buffer de intrare, funcția îl goleşte. În caz de succes returnează valoarea zero, iar în caz de eroare valoarea EOF (definită în *stdio.h*).

Exemplu: Înainte de a citi un şir de caractere de la tastatură, bufferul trebuie golit pentru a preveni citirea unui şir vid (datorită unei perechi CR/LF rămase în buffer de la o citire anterioară a unei valori numerice). Ştergerea se realizează prin apelul:

```
fflush(stdin);
```

Aflarea poziției curente în fişier se realizează prin apelul uneia din funcțiile *fgetpos* sau *ftell*:

```
int fgetpos(FILE* f,fpos_t* poziție);
```

După apel, la adresa *poziție* se află poziția pointerului de citire/scriere din fişierul *f*, ca număr relativ al octetului curent. Primul octet are numărul 0. Valoarea returnată poate fi folosită pentru poziționare cu funcția *fsetpos*. În caz de succes funcția întoarce valoarea 0, iar în caz de eroare, o valoare nenulă şi setează variabila *errno* la valoarea EBADF sau EINVAL.

```
long ftell(FILE* f);
```

returnează poziția în fişierul *f* a pointerului de citire/scriere în caz de succes sau -1L în caz contrar. Dacă fişierul este binar, poziția este dată în număr de octeți față de începutul fişierului. Valoarea poate fi folosită pentru poziționare cu funcția *fseek*.

Modificarea poziției pointerului de citire/scriere se poate face prin poziționare relativă:

```
int fseek(FILE* f,long deplasare,int origine);
```

unde *deplasare* reprezintă numărul de octeți cu care se deplasează pointerul în fişierul *f*, iar *origine* reprezintă poziția față de care se deplasează pointerul. Parametrul *origine* poate fi: SEEK_SET (0) – poziționare față de începutul fişierului; SEEK_CUR (1) – poziționare față de poziția curentă; SEEK_END (2) – poziționare față de sfîrşitul fişierului. Funcția returnează valoarea 0 în caz de succes (şi uneori şi în caz de eşec). Se semnalează eroare prin returnarea unei valori nenule numai în cazul în care *f* nu este deschis.

Poziţionarea absolută se face cu funcţia:

```
int fsetpos(FILE* f,const fpos_t poziţie);
```

Pointerul de citire/scriere se mută în fişierul *f* la octetul cu numărul indicat de parametrul *poziţie* (care poate fi o valoare obţinută prin apelul lui *fgetpos*).

Ambele funcţii resetează indicatorul de sfîrşit de fişier şi anulează efectele unor eventuale apeluri anterioare ale lui *ungetc* asupra acelui fişier.

Redenumirea sau *mutarea* unui fişier existent se poate realiza prin apelul funcţiei *rename*, care are următorul prototip:

```
int rename(const char* n_vechi,const char* n_nou);
```

unde *n_vechi* reprezintă vechiul nume al fişierului, iar *n_nou* reprezintă numele nou. Dacă numele vechi conţine numele discului (de exemplu `c:`), numele nou trebuie să conţină acelaşi nume de disc. Dacă numele vechi conţine o cale, numele nou nu este obligat să conţină aceeaşi cale. Folosind o altă cale se obţine *mutarea* fişierului pe disc. Folosind aceeaşi cale (sau nefolosind calea) se obţine *redenumirea* fişierului. Nu sînt permise *wildcard*-uri (?, *) în cele două nume.

În caz de succes se întoarce valoarea 0. În caz de eroare se întoarce -1 şi *errno* primeşte una din valorile: ENOENT – nu există fişierul, EACCES – nu există permisiunea pentru operaţie sau ENOTSAM – dispozitiv diferit (mutarea se poate face doar pe acelaşi dispozitiv).

Ştergerea unui fişier existent se poate realiza prin apelul funcţiei *unlink*, prezentată anterior, sau *remove*, care are următorul prototip:

```
int remove(const char* cale);
```

unde *cale* reprezintă specificatorul fişierului (trebuie să fie închis).

Funcţii de citire/scriere fără conversie

Funcţiile efectuează transferuri de secvenţe de octeţi între memoria internă şi un fişier de pe disc, fără a interveni asupra conţinutului sau ordinii octeţilor respectivi.

Citirea dintr-un fişier binar se realizează prin apelul funcţiei *fread*, care are următorul prototip:

```
size_t fread(void* ptr,size_t dim,size_t n,FILE* f);
```

Funcţia citeşte din fişierul *f*, de la poziţia curentă, un număr de *n* entităţi, fiecare de dimensiune *dim*, şi le depune, în ordinea citirii, la adresa *ptr*. *fread* returnează numărul de entităţi citite. În total se citesc, în caz de succes, *n***dim* octeţi. În caz de eroare sau cînd se întîlneşte sfîrşitul de fişier, funcţia returnează o valoare negativă

sau 0; s*ize_t* este definit în mai multe *header*-e (între care *stdio.h*) şi este un tip de dată folosit pentru a exprima dimensiunea obiectelor din memorie. Este compatibil cu tipul *unsigned*.

Exemplu:

```
struct complex {int x,y} articol;
FILE * f_complex;
if(f_complex=fopen("NR_COMPL.DAT", "rb")
  fread(&articol,sizeof(articol),1,f_complex);
else printf("Fisierul nu poate fi deschis");
```

În exemplul anterior se deschide un fişier binar din care se citeşte un articol de tip *struct complex* care se depune în variabila *articol*.

Scrierea într-un fişier binar se poate realiza prin apelul funcției *fwrite*, care are următorul prototip:

```
size_t fwrite(const void* ptr,size_t dim,size_t n,FILE* f);
```

Funcția scrie în fişierul *f*, începînd cu poziția curentă, un număr de *n* entități contigue, fiecare de dimensiune *dim*, aflate în memorie la adresa *ptr*; *fwrite* returnează numărul entităților scrise cu succes. În caz de eroare se returnează o valoare negativă.

Exemplu:

```
struct complex {int x,y} articol;
FILE *pf;
pf=fopen("NR_COMPL.DAT","wb");
fwrite(& articol,sizeof (articol),1,pf);
```

Exemplul anterior creează un fişier binar nou în care scrie o secvență de octeți conținînd reprezentarea binară a unei date de tip *struct complex*.

Exemplu:

Să se scrie funcția care calculează numărul de articole dintr-un fişier binar, cunoscînd lungimea în octeți a unui articol. Funcția are ca parametri fişierul şi lungimea în octeți a unui articol. Prin numele funcției se întoarce numărul de articole din fişier.

```
int nrart(FILE *f, int l)
{long p;
 int n;
 p=ftell(f);
 fseek(f,0,2);
 n=ftell(f)/l;
 fseek(f,0,p);
 return n;}
```

Funcţii de citire/scriere cu conversie

Funcţiile efectuează transferuri de secvenţe de octeţi între memoria internă şi un fişier de pe disc, convertind secvenţa de la reprezentarea internă (binară) la reprezentarea externă (ASCII) şi invers.

Transferul de caractere se efectuează prin următoarele funcţii:

```
int fgetc(FILE* f);
int fputc(int c, FILE *f);
int getc(FILE* f);
int putc(int c, FILE *stream);
```

Funcţia *fgetc* şi macrodefiniţia *getc* returnează următorul caracter din fişierul *f* (după ce îl converteşte la reprezentarea de tip întreg fără semn). Dacă s-a ajuns la sfîrşitul fişierului, funcţia va întoarce EOF (valoarea -1). Tot EOF va întoarce şi dacă sînt probleme la citirea din fişier.

Funcţia *fputc* şi macrodefiniţia *putc* scriu caracterul *c* în fişierul *f*. În caz de eroare se returnează valoarea *c*, altfel se returnează EOF.

Funcţia *ungetc* pune caracterul *c* în bufferul de citire asociat fişierului *f*. La următoarea citire cu *fread* sau *getc* acesta va fi primul octet/caracter citit. Un al doilea apel al funcţiei *ungetc*, fără să fie citit primul caracter pus în flux, îl va înlocui pe acesta. Apelarea funcţiilor *fflush*, *fseek*, *fsetpos* sau *rewind* şterge aceste caractere din flux. În caz de succes, *ungetc* returnează caracterul *c*, iar în caz de eroare returnează EOF.

Transferul de şiruri de caractere se efectuează prin funcţiile:

```
char* fgets(char* s,int n,FILE* f);
int fputs(const char* s,FILE* f);
```

Funcţia *fgets* citeşte un şir de caractere din fişierul *f* şi îl depune la adresa s. Transferul se încheie atunci cînd s-au citit *n-1* caractere sau s-a întîlnit caracterul *newline*. La terminarea transferului, se adaugă la sfîrşitul şirului din memorie caracterul nul '\0'. Dacă citirea s-a terminat prin întîlnirea caracterului *newline*, acesta va fi transferat în memorie, caracterul nul fiind adăugat după el (spre deosebire de *gets*, care nu îl reţine). La întîlnirea sîrşitului de fişier (fără a fi transferat vreun caracter) sau în caz de eroare *fgets* returnează *NULL*. În caz de succes returnează adresa şirului citit (aceeaşi cu cea primită în parametrul *s*).

Funcţia *fputs* scrie în fişierul *f* caracterele şirului aflat la adresa *s*. Terminatorul de şir ('\0') nu este scris şi nici nu se adaugă caracterul *newline* (spre deosebire de *puts*). În caz de succes *fputs* returnează ultimul caracter scris. În caz de eroare returnează EOF.

Transferul de date cu format controlat este realizat prin funcţiile:

```
int fprintf(FILE* f,const char* format[,…]);
int fscanf(FILR* f,const char* format[,…]);
```

Cele două funcţii lucrează identic cu *printf* şi *scanf*. Singura diferenţă constă în fişierul în/din care se transferă datele. Dacă *printf* şi *scanf* lucrează cu fişierele standard *stdin* şi *stdoud*, pentru *fprintf* şi *fscanf* este necesară precizarea explicită a fişierului cu care se lucrează, prin parametrul *f*.

Deşi nu lucrează cu fişiere în mod direct, se pot folosi şi funcţiile

```
int sprintf(char *s,const char *format[,...]);
int sscanf(const char *s,const char *format[,...]);
```

Aceste funcţii lucrează identic cu *printf* şi *scanf*, diferenţa constînd în entitatea din/în care se transferă datele. În locul fişierelor standard, acest funcţii folosesc o zonă de memorie de tip şir de caractere, a cărei adresă este furnizată în parametrul *s*. Şirul de la adresa *s* poate fi obţinut prin transfer fără format dintr-un fişier text (pentru *sscanf*) sau poate urma să fie scris într-un fişier text prin funcţia *fputs*.

Pentru *tratarea erorilor* se folosesc următoarele funcţii:

```
void clearerr (FILE* f);
```

Funcţia resetează indicatorii de eroare şi indicatorul de sfîrşit de fişier pentru fişierul *f* (se înscrie valoarea 0). O dată ce indicatorii de eroare au fost setaţi la o valoare diferită de 0, operaţiile de intrare/ieşire vor semnala eroare pînă la apelul lui *clearerr* sau *rewind*.

```
int ferror (FILE* nume_intern);
```

Este o macrodefiniţie care returnează codul de eroare al ultimei operaţii de intrare/ieşire asupra fişierului *nume_intern* (0 dacă nu s-a produs eroare).

Exemplu:

```
#include <stdio.h>
int main(void)
{ FILE *f;
  /* deschide fisierul pentru scriere*/
  f=fopen("test.ttt","w");
  /* se produce eroarela incercarea de citire */
  getc(f);
  if(ferror(f))  /* s-a produs eroare de I/E? */
   {/* afiseaza mesaj de eroare */
    printf("Eroare al citirea din test.ttt\n");
    //reseteazaindicatorii de eroare si sfirsit de fisier
    clearerr(f);}
  fclose(f);
  return 0;}
```

Exemplu:

Să se scrie un program care calculează şi afişează valoarea unei funcţii introduse de la tastatură într-un punct dat. Funcţia se introduce ca şir de caractere şi poate conţine apeluri de funcţii standard C (vezi şi [Smeu95]).

Programul creează un fişier sursă C (în care este scrisă forma funcţiei, ca subprogram C), apoi compilează şi execută un alt program, care va include subprogramul creat. Descrierea funcţiei introduse de la tastatură trebuie să conţină maxim 200 de caractere.

a) Fişierul *51_iii_a.cpp* conţine programul care realizează citirea formei funcţiei, compilarea şi execuţia programului care calculează valoarea funcţiei.

```
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<process.h>
void main()
{ char s1[213]="return(";
  char s2[]="double f(double x)\r\n\{\r\n";
  FILE *f; int n,i,j;
  f=fopen("functie.cpp","w");
  fputs(s2,f);
  printf("functia f(x)="); gets(&s1[7]);
  strncat(s1,");\r\n}",6);
  fputs(s1,f);
  fclose(f);
  system("bcc -Id;\borlandc\include -Ld:\borlandc\lib 51_iii_b.cpp>>
          tmp.txt");
  execl("51_iii_b ",NULL);}
```

b) Fişierul *51_iii_b* conţine programul care citeşte punctul *x*, calculează valoarea funcţiei în acest punct şi o afişează.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include"functie.cpp"
void main()
{double x;
 printf("x=");scanf("%lf",&x);
 printf("f(%7.2lf)=%7.2lf",x,f(x));
 getch();}
```

9.3 Particularităţi ale algoritmilor de prelucrare cu fişier conducător

Caracteristica generală a algoritmilor de prelucrare cu fişier conducător este parcurgerea secvenţială a fişierului conducător şi efectuarea unor prelucrări în funcţie de fiecare articol citit din acesta. Problema care se pune este detectarea sfîrşitului de fişier. Modul în care se realizează acest lucru în Pascal diferă radical de cel din C. În Pascal, funcţia *eof* realiza prima etapă a citirii (transferul datelor

din fişier în buffer) şi de aceea trebuia apelată înainte de citirea efectivă. În C, macrodefiniția *feof* nu face decît să furnizeze valoarea indicatorului de sfîrşit de fişier, care este setat de operația de citire; în program, citirea trebuie să apară înaintea verificării sfârşitului de fişier. Forma generală a algoritmului în cele două limbaje este:

Pascal:

```
while not eof(f) do
begin <citire articol>
      <prelucrare articol citit>
end;
```

C:

```
<citire articol>
while(!feof(f))
{ <prelucrare articol citit>
  <citire articol>}
```

Exemplu:

Crearea şi consultarea unui fişier text care memorează elemente întregi, folosind funcția *feof* pentru gestionarea sfîrşitului de fişier. La crearea fişierului, fişier conducător este fişierul standard de intrare. La afişare, conducător este fişierul *f*.

```
#include<stdio.h>
#include<conio.h>
void main()
{ FILE *f;
  int x; long dim;
  clrscr(); f=fopen("numere.dat","w+");
  scanf("%d",&x);
  while(!feof(stdin))
   {fprintf(f,"%d\n",x); scanf("%d",&x);}
  fseek(f,0,SEEK_SET);
  fscanf(f,"%d",&x);
  while(!feof(f))
   {printf("%d\t",x);
    fscanf(f,"%d",&x);}
  fclose(f);
  getch();}
```

Acelaşi exemplu, folosind fişier binar:

```
#include<stdio.h>
#include<conio.h>
void main()
{ FILE *f;
  int x,g; long dim;
  clrscr(); f=fopen("numere.dat","wb+");
  scanf("%d",&x);
  while(!feof(stdin))
   {fwrite(&x,sizeof(x),1,f);
    scanf("%d",&x);}
  fseek(f,0,SEEK_SET);
  fread(&x,sizeof(x),1,f);
  while(!feof(f))
   {printf("%d\t",x);
    fread(&x,sizeof(x),1,f);}
  fclose(f);
  c=getch();}
```

1. Funcții

i. Să se scrie funcția pentru aproximarea valorii unei integrale, definită prin funcția f(x), pe un interval dat, prin metoda trapezelor.

Funcția are ca parametri de intrare capetele intervalului pe care este definită integrala (*a* şi *b*), numărul de diviziuni ale intervalului (*n*) şi adresa funcției care se integrează (*f*). Funcția returnează, prin numele ei, valoarea aproximativă a integralei. Cu cît numărul de diviziuni este mai mare (lungimea unui subinterval mai mică) cu atît mai bună este aproximarea.

```
double trapez(double a,double b,int n,double (*f)(double))
{ double h,i;
  int j;
  h=(b-a)/n; i=0.0;
  for(j=0;j<=n;j++)
    i+=(*f)(a+j*h);
  i*=h; return i;}
```

ii. Să se scrie funcția pentru determinare celui mai mare divizor comun dintre 2 numere naturale.

Funcția are ca parametri de intrare cele două numere (*a* şi *b*) şi returnează, prin numele ei, valoarea celui mai mare divizor comun.

- varianta recursivă:

```
long cmmdc(long a,long b)
{ long c;
  if(a==b) c=a;
  else if(a>b) c=cmmdc(a-b,b);
       else c=cmmdc(a,b-a);
  return c;}
```

- varianta iterativă:

```
long cmmdc(long a,long b)
{ long r,d=a,i=b;
  do {r=d%i; d=i; i=r;}
  while(r!=0);
  return d;}
```

iii. Să se scrie o funcţie eficientă pentru ridicarea unui număr la o putere naturală.

Funcţia are ca parametri baza (*b*) şi exponentul (*e*) şi returnează, prin numele ei, valoarea cerută.

-varianta iterativă:

```
long putere(int b,int e)
{ long p=1;
  while(e)
    if(e%2) {p*=b;e--;}
    else {b*=b; e/=2;}
  return p;}
```

-varianta recursivă:

```
long putere(int b,int e)
{ long p;
  if(!e) p=1;
  else if(e%2) p=b*putere(b,e-1);
       else p=putere(b,e/2)*putere(b,e/=2);
  return p;}
```

iv. Să se scrie funcţia pentru calcularea sumei elementelor unui masiv tridimensional. Să se folosească diferite variante pentru transmiterea masivului ca parametru.

Funcţia are ca parametri de intrare masivul tridimensional (*a*) şi dimensiunile sale efective (*m*, *n*, *p*). În prima variantă toate cele trei dimensiuni sînt precizate. În a doua variantă numărul de plane este omis (facilitate permisă în C). În a treia variantă se trimite un pointer spre o matrice. Cele trei variante de transmitere a masivului sînt echivalente.

```
int s1(int a[3][3][3],int m,int n,int p)
{ int s=0,i,j,k;
  for(i=0;i<m;i++)
    for(j=0;j<n;j++)
      for(k=0;k<p;k++)
        s+=a[i][j][k];
  return(s);}
int s2(int a[][3][3],int m,int n,int p)
{ int s=0,i,j,k;
  for(i=0;i<m;i++)
    for(j=0;j<n;j++)
      for(k=0;k<p;k++)
        s+=a[i][j][k];
  return(s);}
int s3(int (*a)[3][3],int m,int n,int p)
{ int s=0,i,j,k;
  for(i=0;i<m;i++)
```

```
    for(j=0;j<n;j++)
      for(k=0;k<p;k++)
        s+=a[i][j][k];
  return(s);}
```

v. Să se scrie funcţia pentru afişarea conţinutului binar al unei zone de memorie în care se află memorat un şir de caractere.

Funcţia are ca parametru de intrare adresa şirului de caractere care trebuie afişat şi foloseşte o mască pentru a selecta fiecare bit al fiecărui caracter.

```
void bin(char *s)
{ unsigned char masca;
  while(*s)
   {masca=128;
    while(masca)
     {if(*s&masca)putch('1');
      else putch('0');
      masca>>=1;}
    s++;
    printf("\n");}}
```

vi. Să se scrie funcţia pentru aproximarea valorii soluţiei unei ecuaţii algebrice transcendente prin metoda bisecţiei.

Funcţia are ca parametri de intrare capetele intervalului în care se caută soluţia (*x0* şi *x1*), numărul maxim de iteraţii (*n*), precizia dorită (*eps*), funcţia asociată ecuaţiei (*f*) şi adresa unde se va înscrie soluţia. Prin numele funcţiei se returnează un cod de eroare cu următoarea semnificaţie: 0 – nu s-a găsit soluţie datorită numărului prea mic de iteraţii sau preciziei prea mari cerute; 1 – s-a obţinut soluţia exactă; 2 – s-a obţinut o soluţia aproximativă; 3 – intervalul dat nu conţine nici o soluţie.

-varianta iterativă

```
int bisectie(float x0,float x1,unsigned n,float eps,float
             (*f)(float), float *sol)
{ int cod=0;
  if ((*f)(x0)*(*f)(x1)>0) cod=3;
  else while((n)&&(!cod))
         {*sol=(x0+x1)/2;
          if((*f)(*sol)==0) cod=1;
          if(fabs(x0-x1)<=eps) cod=2;
          else {if((*f)(*sol)*(*f)(x0)<0)x1=*sol;
                else x0=*sol;
                n--;}}
  return cod;}
```

-varianta recursivă

```
int bisectie(float x0,float x1,unsigned n,float eps,float
             (*f)(float),float *sol)
```

```
{ int cod;
  if ((*f)(x0)*(*f)(x1)>0) cod=3;
  else if (n==0) cod=0;
       else {*sol=(x0+x1)/2;
             if((*f)(*sol)==0) cod=1;
             else if(fabs(x0-x1)<=eps) cod=2;
                  else {if((*f)(*sol)*(*f)(x0)<0)
                          cod=bisectie(x0,*sol,n-1,eps,f,sol);
                        else cod=bisectie(*sol,x1,n-1,eps,f,sol);}
  return cod;}
```

vii. Să se scrie funcţia pentru aproximarea valorii soluţiei unei ecuaţii algebrice transcendente prin metoda tangentei.

Funcţia are ca parametri de intrare soluţia iniţială (*x0*), numărul maxim de iteraţii (*n*), precizia cerută (*eps*), valoarea minimă a tangentei (*eps2*), funcţia asociată ecuaţiei (*f*), derivata funcţiei asociate ecuaţiei (*fd*), derivata funcţiei de iteraţie (*gd*) şi adresa unde se va înscrie soluţia. Funcţia returnează prin numele său un cod de eroare cu următoarea semnificaţie: 0 – nu s-a găsit soluţie datorită numărului prea mic de iteraţii; 1 – nu s-a găsit soluţie datorită anulării derivatei funcţiei asociate ecuaţiei; 2 – nu s-a găsit soluţie deoarece metoda nu este convergentă pentru datele primite; 3 – s-a găsit soluţie aproximativă.

-varianta iterativă

```
int tangenta(float x0,int n,float eps,float eps2,float (*f)(float),
             float(*fd)(float),float(*gd)(float),float *x)
{ int cod=0;
  while((n)&&(!cod))
   {if(fabs((*fd)(x0))<eps2) cod=1;
    else if(fabs((*gd)(x0))>1) cod=2;
         else {*x=x0-(*f)(x0)/(*fd)(x0);
               if(fabs(*x-x0)<eps1) cod=3;
               else {x0=*x; n--;}
              }
   }
  return cod;}
```

-varianta recursivă

```
int tangenta(float x0,int n,float eps,float eps2,float (*f)(float),
              float(*fd)(float),float(*gd)(float),float *x)
{ int cod;
  if(n==0) cod=0;
  else {if(fabs((*fd)(x0))<eps2) cod=1;
        else if(fabs((*gd)(x0))>1) cod=2;
             else {*x=x0-(*f)(x0)/(*fd)(x0);
                   if(fabs(*x-x0)<eps1) cod=3;
                   else cod=tangenta(*x,n-1,eps,eps2,f,fd,gd,x);}
       }
  return cod;}
```

viii. Să se scrie funcția pentru calculul lui *n!*, recursiv şi nerecursiv.

Funcția are ca parametru de intrare pe *n* şi returnează, prin numele ei, valoarea factorialului.

-varianta recursivă

```
long fact(long n)
{ long f;
  if (n==1)f=1;
  else f=n*fact(n-1);
  return(f);}
```

-varianta iterativă

```
long fact(long n)
{long f=1;
 for(long i=1;i<=n;i++)
   f*=i;
 return(f);}
```

ix. Să se scrie funcția pentru calcularea termenului de ordin *n* al şirului Fibonacci, recursiv şi nerecursiv.

Funcția are ca parametru de intrare indicele termenului pe care trebuie să îl calculeze şi returnează, prin numele ei, valoarea cerută. Indicii termenilor şirului încep de la 1.

-varianta iterativă

```
long fib(int n)
{ long f,a,b;
  int i;
  if ((n==1)||(n==2))f=1;
  else {a=1;b=1;
        for(i=3;i<=n;i++)
          {f=a+b;
           a=b;b=f;}
       }
  return(f);}
```

-varianta recursivă

```
long fib(int n)
{ long f;
  if ((n==1)||(n==2))f=1;
  else f=fib(n-1)+fib(n-2);
  return(f);}
```

x. Să se scrie un program în care funcţia *main()* afişează parametrii primiţi în linia de comandă.

Funcţia *main()* tratează parametrii din linia de comandă folosind parametrii *argc* şi *argv*. Parametrul *argc* este de tip întreg şi reprezintă numărul parametrilor din linia de comandă plus *1* (primul parametru este considerat a fi numele programului executabil, cu calea completă). Parametrul *argv* este un vector de şiruri de caractere. Fiecare element al vectorului este unul din parametrii primiţi de program, în ordinea primirii lor.

```
#include <stdio.h>
main(int argc, char *argv[])
{ while(argc)
   {printf("\n%s",*argv);
    argc--;
    argv++;}
}
```

xi. Să se scrie o funcţie cu număr variabil de parametri care returnează produsul parametrilor primiţi.

- Numărul parametrilor variabili este transmis ca parametru fix în funcţie:

```
int prod(int n,...)
{ int nr,pr;
  va_list vp;
  va_start(vp,n);
  pr=1;
  for(int i=0;i<n;i++)
   {nr=va_arg(vp,int);
    pr*=nr;}
  va_end(vp);
  return(pr);}
```

-Sfîrşitul listei de parametri este marcat prin transmiterea unei valori convenţionale (*-1* în exemplul următor). Deoarece trebuie să existe cel puţin un parametru fix, primul parametru este adresa unde se va depune rezultatul.

```
int prod(int *pr,...)
{int nr;
  va_list vp;
  va_start(vp,n);
  *pr=1;
  while((nr=va_arg(vp,int))!=-1)
    *pr*=nr;
  va_end(vp);
  return(*pr);}
```

xii. Să se scrie programul care, în funcție de numărul valorilor întregi citite de la tastatură selectează şi lansează automat una din funcțiile:

a. f1=-1, dacă nu se citeşte nici o valoare;
b. f2=$x^{2,}$ dacă se citeşte o valoare;
c. f3=x*y, dacă se citesc 2 valori;
d. f4=x+y+z, dacă se citesc 3 valori;
e. f5=x*y+z*t, dacă se citesc 4 valori.

Selectarea funcțiilor se face verificînd valoarea întoarsă de funcția *scanf*, adică numărul parametrilor corect citiți de la tastatură.

```
#include<stdio.h>
int f1(int a, int b, int c, int d)
{ return -1;}
int f2(int a, int b, int c, int d)
{ return a*a;}
int f3(int a, int b, int c, int d)
{ return a*b;}
int f4(int a, int b, int c, int d)
{ return a+b+c;}
int f5(int a, int b, int c, int d)
{ return a*b+c*d;}
void main()
{ int (*pf)(int, int, int, int);
  int v, x,y,z,t;
  switch(scanf("%d %d %d %d",&x, &y, &z, &t))
  { case 0: pf=f1;break;
    case 1: pf=f2;break;
    case 2: pf=f3;break;
    case 3: pf=f4;break;
    case 4: pf=f5;break;
    default: break;}
  v=(*pf)(x,y,z,t);
  printf("\n Rezultat=%d",v);}
```

xiii. Să se scrie funcția recursivă C pentru rezolvarea problemei turnurilor din Hanoi.

Funcția are ca parametri numărul de discuri (*n)* şi cele trei tije (*a*, *b*, *c*), în ordinea sursă, destinație, intermediar.

```
void Hanoi(unsigned n,unsigned a, unsigned b,unsigned c)
{ if(n>0){
  Hanoi(n-1,a,c,b);
  printf("Transfer disc de pe tija %u pe tija %u\n",a,b);
  Hanoi(n-1,c,b,a);}
}
```

xiv. Scrieţi o funcţie C pentru calculul recursiv al valorii C_n^k.

Funcţia are ca parametri valorile *n* şi *k* şi întoarce, prin numele ei, valoarea C_n^k.

```
long comb(unsigned n, unsigned k)
{ long rez;
  if (k>n) rez=0;
  if ((k==0)||(k=n)) rez=0;
  rez comb(n-1,k)+comb(n-1,k-1);
  return rez;}
```

2. Operații cu masive şi pointeri

2.1 Operații cu vectori

i. Să se scrie funcția pentru citirea unui vector de la tastatură.

Funcția nu are parametri de intrare. Parametrii de ieşire sînt vectorul şi numărul de elemente, pentru care se simulează transferul prin adresă.

```
void citire(int v[],int* n)
{ int i,er;
  printf("\nn=");scanf("%d",n);
  for(i=0;i<*n;i++)
   {printf("v(%d)=",i);
    do {p=scanf("%d",&v[i]);}
    while(p!=1);}}
```

ii. Să se scrie funcția pentru afişarea unui vector pe ecran.

Funcția are ca parametri de intrare vectorul şi numărul de elemente.

```
void afisare(float v[],int n)
{ int i;
  printf("\n"); for(i=0;i<n;i++) printf("\t%5.2f",v[i]);}
```

iii. Să se scrie funcția pentru găsirea elementului minim dintr-un vector.

Funcția are ca parametri vectorul şi numărul de elemente şi returnează, prin numele ei, elementul minim.

```
float minim(float v[],int n)
{ float m; int i;
  m=v[0];
  for(i=0;i<n;i++)
    if(m>v[i])m=v[i];
  return(m);}
```

iv. Să se scrie funcţia pentru găsirea elementului minim şi a primei poziţii de apariţie a acestuia într-un vector.

Funcţia are ca parametri vectorul, numărul de elemente şi adresa unde se va reţine prima poziţie de apariţie a minimului. Prin numele funcţiei se returnează valoarea minimului.

```
float minim(float v[],int n,int *poz)
{ float m;
  m=v[0];
  *poz=0;
  int i;
  for(i=0;i<n;i++)
    if(m>v[i]){m=v[i];
               *poz=i;}
  return(m);}
```

v. Să se scrie funcţia pentru găsirea elementului minim şi a ultimei poziţii de apariţia a acestuia într-un vector.

Funcţia are ca parametri vectorul, numărul de elemente şi adresa unde se va reţine ultima poziţie de apariţie a minimului. Prin numele funcţiei se returnează valoarea minimului.

```
float minim(float v[],int n,int *poz)
{ float m;
  m=v[0];
  *poz=0;
  int i;
  for(i=0;i<n;i++)
    if(m>=v[i]){m=v[i];
                *poz=i;}
  return(m);}
```

vi. Să se scrie funcţia pentru găsirea elementului maxim şi a tuturor poziţiilor sale de apariţie într-un vector.

Funcţia are ca parametri vectorul, numărul de elemente, vectorul unde se vor reţine poziţiile maximului şi adresa unde se va scrie numărul de apariţii ale maximului.

```
float minim(float v[],int n,int poz[],int *nrpoz)
{ float m;
  int i;
  m=v[0]; poz[0]=0;
  *nrpoz=1;
  for(i=1;i<n;i++)
    if(m>v[i]) {m=v[i];
                poz[0]=i;
                *nrpoz=1;}
    else if(m==v[i]) {poz[*nrpoz]=i;
                      (*nrpoz)++;}
  return(m);}
```

vii. Să se scrie funcția pentru inserarea unui 0 între fiecare două elemente ale unui vector.

Funcția are ca parametri vectorul şi adresa numărului de elemente ale sale. La adresa respectivă se va reține noul număr de elemente rezultat în urma prelucrării. Nu se obține nici un rezultat prin numele funcției.

```
void inserare(float v[],int* n)
{ int i,j,k;
  k=*n;
  for(i=0;i<k-1;i++)
   {for(j=*n;j>2*i+1;j--) v[j]=v[j-1];
    v[2*i+1]=0;
    (*n)++;}
}
```

viii. Să se scrie funcția pentru crearea unui vector din elementele unui vector dat, inserînd cîte un 0 între fiecare 2 elemente ale acestuia.

Funcția are ca parametri vectorul inițial şi numărul său de elemente, vectorul rezultat şi adresa unde se va scrie numărul de elemente ale vectorului rezultat. Nu se întoarce nici o valoare prin numele funcției.

```
void inserare(float v[],int n,float v1[],int* n1)
{ int i;
  *n1=0;
  for(i=0;i<n-1;i++)
   {v1[2*i]=v[i];
    v1[2*1+1]=0;
    (*n1)+=2;}
    v1[*n1]=v[n-1];
 (*n1)++;}
```

ix. Să se scrie funcția pentru compactarea unui vector prin eliminarea dublurilor.

Funcția are ca parametri vectorul şi adresa unde se află numărul de elemente ale acestuia. La această adresă se va înscrie numărul de elemente rămase după compactare. Funcția întoarce, prin numele ei, numărul de elemente rămase în vector.

```
int compactare(float v[],int *n)
{ int i,j,k;
  for(i=0;i<*n-1;i++)
    for(j=i+1;j<*n;j++)
      if(v[i]==v[j])
        {for(k=j;k<*n-1;k++)
           v[k]=v[k+1];
         (*n)--;
         j--; }
  return(*n);}
```

x. Să se scrie funcţia pentru crearea unui vector din elementele unui vector dat, fără a lua în considerare dublurile.

Funcţia are ca parametri vectorul, numărul său de elemente, vectorul care se va construi, adresa unde se va înscrie numărul de elemente ale vectorului rezultat. Nu se întoarce nici o valoare prin numele funcţiei.

```
void compactare(float v[],int n,float v1[],int *n1)
{ int i,j,k;
  *n1=0;
  for(i=0;i<n;i++)
    {k=0;
     for(j=0;j<*n1;j++)
       if(v[i]==v1[j]) k=1;
     if(!k)
      {v1[*n1]=v[i];
       (*n1)++;}
   }
}
```

xi. Să se scrie funcţia pentru inversarea ordinii elementelor unui vector.

Funcţia are ca parametri vectorul şi numărul său de elemente. Nu se întoarce nici un rezultat prin numele funcţiei.

```
void inversare(float v[],int n)
{ int i, j;
  float a;
  i=0; j=n-1;
  while(i<j)
  {a=v[i];
   v[i]=v[j];
   v[j]=a;
   i++;
   j--;}
}
```

xii. Să se scrie funcţia pentru calcularea amplitudinii elementelor unui vector.

Funcţia are ca parametri vectorul şi numărul de elemente şi întoarce, prin numele ei, amplitudinea elementelor.

```
float amplitudine(float v[],int n)
{ int i;
  float min,max;
  min=v[0];
  max=v[0];
  for(i=0;i<n;i++)
    if(v[i]<min) min=v[i];
    else if(v[i]>max) max=v[i];
  return(max-min);}
```

xiii. Să se scrie funcția pentru calcularea mediei aritmetice a elementelor unui vector.

Funcția are ca parametri vectorul și numărul de elemente și întoarce, prin numele ei, media aritmetică a elementelor.

```
float mediaa(float v[],int n)
{ int i;
  float s;
  s=0;
  for(i=0;i<n;i++)
    s+=v[i];
  return(s/n);
}
```

xiv. Să se scrie funcția pentru calcularea mediei armonice a elementelor nenule ale unui vector.

Funcția are ca parametri vectorul, numărul de elemente și adresa unde va scrie parametrul de eroare și întoarce, prin numele ei, media armonică a elementelor nenule. Parametrul de eroare este 1 dacă nu se poate calcula media și 0 în caz contrar.

```
float mediaarm(float v[],int n,int *er)
{ int i,m;
  float s;
  s=0;m=0;*er=0;
  for(i=0;i<n;i++)
    if(v[i]!=0){ s+=1/v[i];
                 m++;}
  if(s=0) *er=1;
  else s=m/s;
  return(s);
}
```

xv. Să se scrie funcția pentru calcularea abaterii medii pătratice a elementelor unui vector.

Funcția are ca parametri vectorul și numărul de elemente și apelează funcția pentru calculul mediei aritmetice a elementelor vectorului (descrisă la problema *xiii*). Valoarea abaterii medii pătratice este returnată prin numele funcției.

```
float abatere(float v[],int n)
{float m,s;
 int i;
 m=mediaa(v,n);
 s=0;
 for(i=0;i<n;i++)
   s+=(v[i]-m)*(v[i]-m);
 return(s);
}
```

xvi. Să se scrie funcţiile pentru sortarea unui vector folosind algoritmii:

a. metoda bulelor;
b. metoda selecţiei;
c. sortare rapidă;
d. sortare prin interclasare.

Funcţiile au ca parametri vectorul şi numărul de elemente. Nu se întoarce nici un rezultat prin numele funcţiei.

a)

```
void bule(float v[],int n)
{ int i,p;
  float a;
  p=1;
  while(p)
    {p=0;
     for(i=0;i<n-1;i++)
       if(v[i]>v[i+1])
         {a=v[i];
          v[i]=v[i+1];
          v[i+1]=a;
          p=1; }
    }
}
```

b)

```
void selectie(float v[],int n)
{ float a;
  int i,j,p;
  for(i=0;i<n-1;i++)
    {p=i;
     for(j=i;j<n;j++)
       if(v[p]>v[j])p=j;
     a=v[p];
     v[p]=v[i];
     v[i]=a;}
}
```

c)

```
void quicksort(float v[],int inf,int sup)
{ int i,j,ii,jj;
  float a;
  if(inf<sup)
    {i=inf;j=sup;ii=0;jj=-1;
     while(i<j)
       {if(v[i]>v[j])
          {a=v[i]; v[i]=v[j]; v[j]=a;
           if(ii==0){ii=1;jj=0;}
        else{ii=0;jj=-1;}
       }
     i+=ii;j+=jj;}
     quicksort(v,inf,i-1);
     quicksort(v,i+1,sup);
    }
}
```

```
d)
void interclasare(float v[],int a,int b,int c,int d)
{ int i,j,k;
  float v1[100];
  i=a;j=c;k=0;
  while((i<=b)&&(j<=d))
    if(v[i]<v[j])  v1[k++]=v[i++];
    else           v1[k++]=v[j++];
  if(i>b) for(i=j;i<=d;i++) v1[k++]=v[i];
  else    for(j=i;j<=b;j++) v1[k++]=v[j];
  for(i=0;i<k;i++)
    v[a+i]=v1[i];}

void sort_int(float v[],int s,int d)
{ int m;
  float a;
  if(d-s<2)
   {if(v[s]>v[d]){a=v[s]; v[s]=v[d]; v[d]=a;}}
  else {m=(d+s)/2;
        sort_int(v,s,m);
        sort_int(v,m+1,d);
        interclasare(v,s,m,m+1,d);}
}
```

xvii. Să se scrie funcția pentru interclasarea elementelor a doi vectori sortați crescător.

Funcția are ca parametri primul vector, numărul său de elemente, al doilea vector, numărul său de elemente, vectorul în care va scrie rezultatul și adresa la care va scrie numărul de elemente din vectorul rezultat. Nu se întoarce nici un rezultat prin numele funcției.

```
void interclasare(float v[],int m,float w[],int n,float r[],int* p)
{int i,j;
 i=0;j=0;*p=0;
 while((i<m)&&(j<n))
   if(v[i]<w[j])
     r[(*p)++]=v[i++];
   else
     r[(*p)++]=w[j++];
 if(i==m) for(i=j;i<n;i++)
            r[(*p)++]=w[i];
 else     for(j=i;j<m;j++)
            r[(*p)++]=v[j];
}
```

xviii. Să se scrie funcția pentru calcularea produsului scalar dintre doi vectori.

Funcția are ca parametri cei doi vectori și numărul de elemente ale fiecăruia și adresa unde va scrie parametrul de eroare. Prin numele funcției se întoarce produsul scalar. Parametrul de eroare are valoarea 0, dacă se calculează produsul, sau 1, dacă vectorii au lungimi diferite.

```
float prod_scal(float v[],int n,float v1[],int n1,int *er)
{ float p;
  int i;
  if(n1!=n)*er=1;
  else{*er=0;
       p=0;
       for(i=0;i<n;i++)
         p+=v[i]*v1[i];}
  return(p);}
```

xix. Să se scrie funcția pentru calcularea produsului vectorial dintre doi vectori.

Funcția are ca parametri cei doi vectori, numărul de elemente ale fiecăruia şi vectorul în care va scrie rezultatul. Prin numele funcției se întoarce parametrul de eroare. Parametrul de eroare are valoarea 0, dacă se calculează produsul, sau 1, dacă vectorii au lungimi diferite.

```
int prod_vect(float v[],int n,float v1[],int n1,float r[])
{ int i,er;
  if(n1!=n)er=1;
  else{er=0;
       for(i=0;i<n;i++)
       r[i]=v[i]*v1[i];}
  return(er);}
```

xx. Să se scrie funcția pentru căutarea unui element într-un vector nesortat.

Funcția are ca parametri vectorul, numărul de elemente şi valoarea căutată. Prin numele funcției se întoarce poziția primei apariții a elementului în vector sau –1 dacă elementul nu este găsit.

```
int cautare(float v[],int n,float x)
{ int i,er;
  er=-1;
  for(i=0;i<n;i++)
    if((v[i]==x)&&(er==-1)) er=i;
  return(er);}
```

xxi. Să se scrie funcția pentru căutarea unui element într-un vector sortat.

a) *Varianta iterativă*: funcția are ca parametri vectorul, numărul de elemente şi valoarea căutată. Prin numele funcției se întoarce poziția elementului găsit sau –1, dacă elementul nu a fost găsit.

```
int cautare_bin(float v[],int n,float x)
{ int i,j,er,p;
  er=-1;
  i=0;j=n-1;
  while((i<=j)&&(er==-1))
   {p=(i+j)/2;
    if(v[p]==x) er=p;
    else if(v[p]<x)i=p+1;
         else j=p-1;}
 return(er);}
```

b) *Varianta recursivă*: funcția are ca parametri vectorul, capetele intervalului în care face căutarea (inițial *0* și *n-1*) și valoarea căutată. Prin numele funcției se întoarce poziția elementului găsit sau –1, dacă elementul nu a fost găsit.

```
int cautare_bin_r(float v[],int s,int d,float x)
{ int i,j,er,p;
  p=(s+d)/2;
  if(s>d)er=-1;
  else if(v[p]==x) er=p;
       else if(v[p]<x) er=cautare_bin_r(v,p+1,d,x);
            else       er=cautare_bin_r(v,s,p-1,x);
  return(er);}
```

xxii. Să se scrie funcția pentru determinarea numerelor naturale prime mai mici decît o valoare dată (maxim 1000) prin metoda ciurului lui Eratostene.

Funcția are ca parametri limita maximă, vectorul în care va scrie numerele prime mai mici decît acea limită și adresa unde va scrie numărul de numere găsite. Parametrul de eroare (1 dacă limita este mai mare de 1000, 0 dacă nu sînt erori) este întors prin numele funcției.

```
int eratostene(int x,int v[],int *y)
{ int i,er,q,v1[1000];
  if(x>500)er=1;
  else{er=0;
       for(i=0;i<x;i++) v1[i]=i;
         for(i=2;i<=sqrt(x);i++)
           {q=2*i;
            while(q<x)
             {v1[q]=0; q+=i;}
           }
       *y=0;
       for(i=0;i<x;i++)
         if(v1[i]) v[(*y)++]=v1[i];}
  return er;}
```

xxiii. Să se scrie funcția pentru determinarea valorii unui polinom într-un punct dat.

Funcția are ca parametri gradul polinomului *n*, vectorul coeficienților *a* (în ordine, primul coeficient fiind cel al termenului liber, în total *n+1* elemente) și punctul în care se calculează valoarea polinomului. Prin numele funcției se întoarce valoarea calculată.

```
float polinom(int n,float a[],float x)
{ int i;
  float p;
  p=a[n];
  for(i=n;i>0;i--)
    p=p*x+a[i-1];
  return p;}
```

xxiv. Să se scrie funcţia pentru calculul sumei a două polinoame.

Funcţia are ca parametri gradul primului polinom şi vectorul coeficienţilor săi, gradul celui de al doilea polinom şi vectorul coeficienţilor săi, vectorul în care se vor scrie coeficienţii polinomului rezultat şi adresa la care se va scrie gradul polinomului rezultat.

```
void s_polinom(int n,float a[],int m,float b[],float r[],int* p)
{ int i;
  *p=m>n?m:n;
  for(i=0;i<=*p;i++)
    r[i]=(i>n?0:a[i])+(i>m?0:b[i]);}
```

xxv. Să se scrie funcţia pentru calcul produsului dintre două polinoame.

Funcţia are ca parametri gradul primului polinom, vectorul cu coeficienţii săi, gradul celui de al doilea polinom, vectorul cu coeficienţii săi, vectorul în care se vor scrie coeficienţii polinomului rezultat şi adresa la care se va scrie gradul polinomului rezultat.

```
void p_polinom(int n,float a[],int m,float b[],float r[],int* p)
{ int i,j,tt;
  float t[100];
  *p=0; tt=0;
  for(i=0;i<=n;i++)
   {tt=m+i;
    for(j=0;j<=m;j++)  t[j+i]=b[j]*a[i];
    for(j=0;j<i;j++) t[j]=0;
    s_polinom(tt,t,*p,r,r,p);
    getch();}
}
```

2.2 Probleme cu numere întregi foarte mari

Numerele foarte mari vor fi reprezentate în vectori astfel: fiecare cifră în cîte un element al vectorului, pe poziţia egală cu rangul cifrei respective (cifra unităţilor pe poziţia 0, cifra zecilor pe poziţia 1 etc.) Semnul se va reţine separat sub formă de caracter. Separat se va reţine lungimea numărului (numărul de cifre). Pentru memorarea unui număr foarte mare se va folosi structura *NR* care poate memora un număr cu maxim 500 de cifre:

```
typedef struct{unsigned char x[501];
               int n;
               char s}NR;
```

i. Să se scrie funcţia pentru citirea de la tastatură a unui număr întreg foarte mare, cu semn.

Funcția are ca parametru adresa unde se va memora numărul citit şi întoarce, prin numele ei, numărul de cifre ale acestuia (0 va indica faptul că nu s-a citit nimic). Se citeşte întîi semnul (+/-) şi apoi cifrele, începînd cu cea de rang maxim, pînă la apăsarea altei taste decît o cifră.

```
int citire(NR* a)
{ char c;int i;
  unsigned char t[500];
  printf("\nnumarul cu semn=");
  do a->s=getch();
  while((a->s!='+')&&(a->s!='-'));
  putch(a->s);
  a->n=0;
  do{c=getche();
     t[a->n]=c-'0';
     a->n++;}
  while((c>='0')&&(c<='9'));
  a->n--;
  for(i=0;i<a->n;i++)
    a->x[i]=t[a->n-i-1];
  return a->n;}
```

ii. Să se scrie funcția pentru afişarea pe ecran a unui număr foarte mare, cu semn.

Funcția are ca parametru numărul care trebuie afişat şi nu returnează nici o valoare.

```
void afisare(NR a)
{ int i;
  printf("\n"); putch(a.s);
  if(a.n==0)putch('0');
  else  for(i=0;i<a.n;i++) printf("%d",a.x[a.n-i-1]);}
```

iii. Să se scrie funcția pentru calculul sumei a două numere foarte mari de acelaşi semn.

Funcția are ca parametri cele două numere care se adună şi adresa la care va scrie numărul rezultat. Nu se întoarce nici o valoare prin numele funcției.

```
void suma(NR a,NR b,NR* c)
{int i,max;
 unsigned char t,s;
 max=(a.n>b.n)?a.n:b.n;
 c->s=a.s;
 t=0;
 for(i=0;i<max;i++)
   {s=(i<a.n?a.x[i]:0)+(i<b.n?b.x[i]:0)+t;
    c->x[i]=s%10;
    t=s/10;}
 if(t!=0) {c->x[max]=t;c->n=max+1;}
 else c->n=max;}
```

iv. Să se scrie funcţia pentru înmulţirea unui număr foarte mare cu 10^p.

Funcţia are ca parametri numărul, puterea lui 10 (p) şi adresa unde se va scrie numărul rezultat. Nu se întoarce nici o valoare prin numele funcţiei.

```
void prod_10(NR a,unsigned char p,NR* b)
{ int i;
  for(i=0;i<a.n;i++)
    b->x[i+p]=a.x[i];
  for(i=0;i<p;i++)
    b->x[i]=0;
  b->n=p+a.n;
  b->s=a.s;}
```

v. Să se scrie funcţia pentru calculul produsului dintre un număr foarte mare şi o cifră.

Funcţia are ca parametri numărul foarte mare, cifra şi adresa unde va scrie numărul rezultat. Nu se întoarce nici o valoare prin numele funcţiei.

```
void prod_c(NR a,unsigned char c,NR* b)
{ int i;
  unsigned char t,s;
  b->s=a.s;
  t=0;
  for(i=0;i<a.n;i++)
   {s=a.x[i]*c+t;
    b->x[i]=s%10;
    t=s/10;}
  if(t!=0) {b->x[a.n]=t;
            b->n=a.n+1;}
  else b->n=a.n;}
```

vi. Să se scrie funcţia pentru calculul produsului dintre două numere foarte mari.

Funcţia are ca parametri cele două numere care se înmulţesc şi adresa unde va scrie rezultatul. Nu se întoarce nici o valoare prin numele funcţiei.

```
void prod(NR a,NR b,NR* c)
{ int i;
  NR t;
  if(a.s==b.s) c->s='+';
  else c->s='-';
  c->n=0;
  for(i=0;i<a.n;i++)
   {t.n=0;
    t.s=b.s;
    prod_c(b,a.x[i],&t);
    prod_10(t,i,&t);
    suma(*c,t,c);}
}
```

vii. Să se scrie funcția pentru determinarea maximului în valoare absolută dintre două numere foarte mari.

Funcția are ca parametri cele două numere şi întoarce, prin numele ei, o valoare negativă dacă primul număr e mai mic, zero dacă cele două numere sînt egale sau o valoare pozitivă dacă primul număr este mai mare.

```
int max(NR a,NR b)
{ int i,m;
  m=a.n<b.n?-1:(a.n==b.n?0:1);
  i=a.n;
  while((m==0)&&(i>=0))
   {m=a.x[i]-b.x[i];
    i--;}
  return m;}
```

viii. Să se scrie funcția pentru calculul diferenței a două numere foarte mari, de același semn.

Funcția are ca parametri cele două numere şi adresa unde va scrie numărul rezultat. Este folosită funcția *copiere* pentru a realiza atribuirea între două numere foarte mari; primul parametru este cel care dă valoare, al doilea este cel care primește valoarea.

```
void copiere(NR a,NR *b)
{int i;
 b->s=a.s;
 b->n=a.n;
 for(i=0;i<a.n;i++)
   b->x[i]=a.x[i];}

void diferenta(NR a,NR b,NR *c)
{ int i,m,impr;
  NR d;
  m=max(a,b);
  if(m>0)c->s=a.s;
  else if(m<0)c->s=(a.s=='+')?'-':'+';
       else {m=0;c->n=0;c->s='+';}
  if(m!=0)
   {if(m<0){copiere(a,&d);
            copiere(b,&a);
            copiere(d,&b);}
    impr=0;
    for(i=0;i<a.n;i++)
      {c->x[i]=a.x[i]-(i<b.n?b.x[i]:0)-impr;
       if(c->x[i]<0){c->x[i]+=10;impr=0;}
       else impr=0;}
    c->n=a.n;
    while(c->x[c->n-1]==0) c->n--;}
}
```

2.3 Mulţimi reprezentate ca vectori

Pentru reprezentarea unei mulţimi se va folosi următoarea structură:

```
typedef struct{int x[100];
               int n;}MULTIME;
```

Vectorul *x* conţine elementele mulţimii (maxim 100) iar *n* este numărul de elemente.

i. Să se scrie funcţia pentru căutarea unui element într-o mulţime.

Funcţia are ca parametri mulţimea în care caută şi elementul căutat. Prin numele funcţiei se întoarce valoarea 1 dacă elementul a fost găsit sau 0 dacă nu a fost găsit.

```
int cautare(MULTIME a,int x)
{ int i,er;
  er=0;
  for(i=0;i<a.n;i++)
    if(a.x[i]==x) er=1;
  return er;}
```

ii. Să se scrie funcţia pentru calculul intersecţiei dintre două mulţimi.

Funcţia are ca parametri cele două mulţimi şi adresa unde se va scrie mulţimea rezultat.

```
void diferenta(MULTIME a,MULTIME b,MULTIME *c)
{ int i,j,k;
  c->n=0;
  for(i=0;i<a.n;i++)
    if(cautare(b,a.x[i]))
       c->x[c->n++]=a.x[i];}
```

iii. Să se scrie funcţia pentru calculul reuniunii dintre două mulţimi.

Funcţia are ca parametri cele două mulţimi şi adresa unde se va scrie mulţimea rezultat.

```
void reuniune(MULTIME a, MULTIME b, MULTIME *c)
{ int i,j,k;
  c->n=a.n;
  for(i=0;i<a.n;i++)
    c->x[i]=a.x[i];
  for(i=0;i<b.n;i++)
    if(!cautare(*c,b.x[i]))
      c->x[c->n++]=b.x[i];}
```

iv. Să se scrie funcția pentru calculul diferenței dintre două mulțimi.

Funcția are ca parametri cele două mulțimi şi adresa unde se va scrie mulțimea rezultat.

```
void diferenta(MULTIME a,MULTIME b,MULTIME *c)
{ int i,j,k;
  c->n=0;
  for(i=0;i<a.n;i++)
    if(!cautare(b,a.x[i]))
      c->x[c->n++]=a.x[i];}
```

v. Să se scrie funcția pentru calculul diferenței simetrice dintre două mulțimi.

Funcția are ca parametri cele două mulțimi şi adresa unde se va scrie mulțimea rezultat.

```
int diferenta_simetrica(MULTIME a, MULTIME b,MULTIME *c)
{ MULTIME d,d1;
  diferenta(a,b,d);
  diferenta(b,a,d1);
  reuniune(d,d1,c);}
```

2.4 Operații cu matrice

i. Să se scrie funcția pentru citirea unei matrice de la tastatură.

Funcția are ca parametri adresa unde se vor scrie elementele matricei, adresa unde se va scrie numărul de linii şi adresa unde se va scrie numărul de coloane.

```
void citire(float a[][20],int *m,int *n)
{ int i,j;
  printf("nr linii=");scanf("%d",m);
  printf("nr linii=");scanf("%d",n);
  for(i=0;i<*m;i++)
    for(j=0;j<*n;j++)
     {printf("a(%d,%d)=",i,j);
      scanf("%f",&a[i][j]);}
}
```

ii. Să se scrie funcția pentru afişarea unei matrice pe ecran.

Funcția are ca parametri matricea, numărul de linii şi numărul de coloane. Nu se întoarce nici un rezultat prin numele funcției.

```
void afisare(float a[][20],int m,int n)
{ int i,j;
  printf("\n");
  for(i=0;i<m;i++)
```

```
   {for(j=0;j<n;j++)
      printf("\t%4.2f",a[i][j]);
    printf("\n");}
}
```

iii. Să se scrie funcţia pentru găsirea elementului minim dintr-o matrice.

Funcţia are ca parametri matricea, numărul de linii şi numărul de coloane şi întoarce, prin numele ei, elementul minim.

```
float minim(float a[][20],int m,int n)
{ int i,j;
  float min;
  min=a[0][0];
  for(i=0;i<m;i++)
     for(j=0;j<n;j++)
       if(a[i][j]<min)min=a[i][j];
  return min;b}
```

iv. Să se scrie funcţia pentru găsirea elementelor minime de pe diagonalele principală şi, respectiv, secundară ale unei matrice.

Funcţia are ca parametri matricea, numărul liniilor, numărul coloanelor, adresa unde va scrie elementul minim de pe diagonala principală şi adresa unde va scrie elementul minim de pe diagonala secundară. Prin numele funcţiei se întoarce valoarea 1, dacă matricea nu a fost pătrată, sau 0 în caz contrar.

```
int minim(float a[][20],int m,int n,float *m1,float *m2)
{ int i,er;
  if(m!=n)er=1;
  else{*m1=a[0][0];*m2=a[0][m-i-1];
       for(i=0;i<m;i++)
        {if(a[i][i]<*m1)*m1=a[i][i];
         if(a[i][m-i-1]<*m2)*m2=a[i][m-i-1];}
       er=0;}
  return er;}
```

v. Să se scrie funcţia pentru găsirea elementului maxim din triunghiul de deasupra diagonalelor unei matrice pătrate (exclusiv diagonalele).

Funcţia are ca parametri matricea şi dimensiunea ei şi întoarce, prin numele ei, valoarea cerută.

```
float maxim1(float a[][20],int m)
{ int i,j;float max;
  max=a[0][1];
  for(i=0;i<(m-1)/2;i++)
    for(j=i+1;j<m-i-1;j++)
     if(a[i][j]>max)max=a[i][j];
  return max;}
```

vi. Să se scrie funcția pentru găsirea elementului maxim din triunghiul de sub diagonalele unei matrice pătrate (exclusiv diagonalele).

Funcția are ca parametri matricea şi dimensiunea ei şi întoarce, prin numele ei, valoarea cerută.

```
float maxim2(float a[][20],int m)
{ int i,j;float max;
  max=a[m-1][1];
  for(i=m/2+1;i<m;i++)
    for(j=m-i-1;j<i;j++)
      if(a[i][j]>max)max=a[i][j];
  return max;}
```

vii. Să se scrie funcția pentru găsire elementului minim din triunghiul din stînga diagonalelor unei matrice pătrate (exclusiv diagonalele).

Funcția are ca parametri matricea şi dimensiunea ei şi întoarce, prin numele ei, valoarea cerută.

```
float maxim3(float a[][20],int m)
{ int i,j;
  float max;
  max=a[1][0];
  for(i=0;i<(m-1)/2;i++)
    for(j=i+1;j<m-i-1;j++)
      if(a[j][i]>max)max=a[j][i];
  return max;}
```

viii. Să se scrie funcția pentru găsirea elementului minim din triunghiul din dreapta diagonalelor unei matrice pătrate (exclusiv diagonalele).

Funcția are ca parametri matricea şi dimensiunea ei şi întoarce, prin numele ei, valoarea cerută.

```
float maxim4(float a[][20],int m)
{ int i,j;
  float max;
  max=a[1][m-1];
  for(i=m/2+1;i<m;i++)
    for(j=m-i-1;j<i;j++)
      if(a[j][i]>max)max=a[j][i];
  return max;}
```

ix. Să se scrie secvența de program pentru inițializarea unei matrice la declarare astfel:

a. Elementele de pe prima linie cu valoarea 1, restul cu valoarea 0.
b. Elementele de pe prima coloană cu valoarea 1, restul cu valoarea 0.
c. Elementele din triunghiul de sub diagonala principală (inclusiv diagonala) cu valoarea 1, restul cu valoarea 0.

```
a) float a[5][5]={1,1,1,1,1};
b) float a[5][5]={{1},{1},{1},{1},{1}};
c) float a[5][5]={{1},{1,1},{1,1,1},{1,1,1,1},{1,1,1,1,1}};
```

x. Să se scrie funcţia pentru calculare produsului dintre 2 matrice.

Funcţia are ca parametri, în ordine: matricea deînmulţit, numărul de linii şi numărul de coloane ale acesteia, matricea înmulţitor, numărul de linii şi numărul de coloane ale acesteia, adresa unde se va scrie matricea rezultat, adresele unde se vor scrie numărul de linii şi numărul de coloane ale matricei rezultat. Prin numele funcţiei se întoarce valoarea 1 dacă înmulţirea nu este posibilă sau 0 în caz contrar.

```
int produs(float a[][20],int m,int n,float b[][20],int p,int q,
           float c[][20],int *r,int *s)
{ int er,i,j,k;
  if(n!=p)er=1;
  else{er=0;
       *r=m;*s=q;
       for(i=0;i<*r;i++)
       for(j=0;j<*s;j++)
        {c[i][j]=0;
         for(k=0;k<n;k++)
           c[i][j]+=a[i][k]*b[k][j];}
        }
 return er;}
```

xi. Să se scrie funcţia pentru calcularea produsului dintre 2 matrice binare.

Funcţia are ca parametri, în ordine: matricea deînmulţit, numărul de linii şi numărul de coloane ale acesteia, matricea înmulţitor, numărul de linii şi numărul de coloane ale acesteia, adresa unde se va scrie matricea rezultat, adresele unde se vor scrie numărul de linii şi numărul de coloane ale matricei rezultat. Prin numele funcţiei se întoarce valoarea 1 dacă înmulţirea nu este posibilă sau 0 în caz contrar.

```
int produs(unsigned char a[][20],int m,int n,unsigned char b[][20],
           int p,int q,unsigned char c[][20],int *r, int *s)
{ int er,i,j,k;
  if(n!=p)er=1;
  else{er=0;
       *r=m;*s=q;
       for(i=0;i<*r;i++)
         for(j=0;j<*s;j++)
          {c[i][j]=0;
           for(k=0;k<n;k++)
             c[i][j]=c[i][j]||(a[i][k]&&b[k][j]);}
          }
  return er;}
```

xii. Să se scrie funcţia pentru calcularea produsului dintre o matrice şi un vector.

Funcţia are ca parametri, în ordine: matricea deînmulţit, numărul de linii şi numărul de coloane ale acesteia, vectorul înmulţitor, adresa unde se va scrie vectorul rezultat şi adresa unde se va scrie numărul de elemente ale vectorului rezultat. Prin numele funcţiei se întoarce valoarea 1 dacă înmulţirea nu este posibilă sau 0 în caz contrar.

int produs(float a[][20],int m,int n,float b[],int p,float c[], int *r)

```
{ int er,i,j,k;
  if(n!=p)er=1;
  else{er=0;
       *r=m;
       for(i=0;i<*r;i++)
         {c[i]=0;
          for(k=0;k<n;k++)
            c[i]+=a[i][k]*b[k];}
         }
  return er;}
```

xiii. Să se scrie funcția pentru calcularea produsului dintre un vector şi o matrice.

Funcția are ca parametri, în ordine: vectorul deînmulțit, numărul său de elemente, matricea înmulțitor, numărul de linii şi numărul de coloane ale acesteia, adresa unde se va scrie vectorul rezultat, adresa unde se va scrie numărul de elemente ale vectorului rezultat. Prin numele funcției se întoarce valoarea 1 dacă înmulțirea nu este posibilă sau 0 în caz contrar.

```
int produs(float a[],int m,float b[][20],int n,int p,float c[],
           int *r)
{ int er,i,j,k;
  if(m!=n)er=1;
  else{er=0;
       *r=p;
       for(i=0;i<*r;i++)
        {c[i]=0;
         for(k=0;k<n;k++)
         c[i]+=a[k]*b[k][i];}
        }
  return er;}
```

xiv. Să se scrie funcția pentru sortarea primei linii a unei matrice fără a schimba structura coloanelor.

Funcția are ca parametri matricea şi dimensiunile sale (număr de linii şi număr de coloane). Prin numele funcției nu se întoarce nici o valoare.

```
void sortare(float a[][20],int m,int n)
{ int i,j,k;
  float aux;
  k=1;
  while(k)
   {k=0;
    for(i=0;i<n-1;i++)
      if(a[0][i]>a[0][i+1])
        for(j=0;j<m;j++)
         {aux=a[j][i];
          a[j][i]=a[j][i+1];
          a[j][i+1]=aux;
          k=1;} } }
```

xv. Să se scrie funcţia pentru determinarea liniilor unei matrice care au elementele în ordine strict crescătoare.

Funcţia are ca parametri matricea, numărul de linii şi numărul de coloane ale acesteia, adresa (vectorul) unde va scrie numerele liniilor care îndeplinesc condiţia şi adresa unde va scrie numărul de linii care îndeplinesc condiţia. Nu se întoarce nici un rezultat prin numele funcţiei.

```
void cresc(float a[][20],int m,int n,int b[],int *p)
{ int i,j,k;
  *p=0;
  for(i=0;i<m;i++)
   {k=1;
    for(j=0;j<n-1;j++)
      if(a[i][j]>=a[i][j+1])k=0;
    if(k)b[(*p)++]=i;}
}
```

xvi. Să se scrie funcţia pentru determinarea coloanelor cu toate elementele nule.

Funcţia are ca parametri matricea, numărul de linii şi numărul de coloane ale acesteia, adresa (vectorul) unde va scrie numerele coloanelor care îndeplinesc condiţia şi adresa unde va scrie numărul de coloane care îndeplinesc condiţia. Nu se întoarce nici un rezultat prin numele funcţiei

```
void nule(float a[][20],int m,int n,int b[],int *p)
{ int i,j,k;
  *p=0;
  for(i=0;i<n;i++)
   {k=1;
    for(j=0;j<m;j++)
      if(!a[j][i])k=0;
    if(k)b[(*p)++]=i;}
}
```

xvii. Să se scrie funcţia pentru construirea unei noi matrice cu liniile şi coloanele unei matrice care nu conţin o anumită valoare.

Funcţia are ca parametri matricea iniţială şi dimensiunile sale (numărul linii-lor şi numărul coloanelor), noua matrice (adresa unde vor fi scrise elementele sale), adresele unde se vor scrie dimensiunile sale, numărul liniilor şi numărul coloanelor. Sînt apelate funcţiile *compactare* (exerciţiul 2.1.ix) şi *căutare* (exerciţiul 2.1.xx).

```
void nenule(float a[][20],int m,int n,float b[][20], int *p,int *q)
{ int i,j,k,r,s,l[20],c[20];
  k=0;
  r=0;
  for(i=0;i<m;i++)
    for(j=0;j<n;j++)
      if(!a[j][i])
```

```
      {l[k++]=i;
       c[r++]=j;}
  compactare(l,&k);
  compactare(c,&r);
  *p=0;*q=n;
  for(i=0;i<m;i++)
    if(cautare(l,k,i)==-1)
      {for(j=0;j<n;j++)
         b[*p][j]=a[i][j];
       (*p)++;}
  for(j=0;j<n;j++)
    if(cautare(c,r,j)!=-1)
      {for(s=j;s<n-1;s++)
         for(i=0;i<*p;i++)
           b[i][s]=b[i][s+1];
       n--;}
  *q=n;}
```

xviii. Să se scrie funcția pentru ridicarea la putere a unei matrice pătrate.

Funcția are ca parametri matricea inițială, dimensiunea ei, puterea la care se ridică și matricea în care va scrie rezultatul. Sînt folosite funcțiile *copiere* (pentru copierea unei matrice în alta) și *produs* (pentru înmulțirea a două matrice – exercițiul 2.4.x).

```
void copiere(float a[][20],int m, float b[][20])
{ int i,j;
  for(i=0;i<m;i++)
    for(j=0;j<m;j++)
      b[i][j]=a[i][j];}

void putere(float a[][20],int m, int p,float b[][20])
{ int i,j,k;
  float c[20][20];
  for(i=0;i<m;i++)
    for(j=0;j<m;j++)
      c[i][j]=(i==j);
  for(i=0;i<p;i++)
   {produs(c,m,m,a,m,m,b,&m,&m);
    copiere(b,m,c);}
}
```

xix. Să se scrie funcția pentru rezolvarea unui sistem algebric liniar de *n* ecuații cu *n* necunoscute, calculînd în acelaşi timp inversa matricei sistemului şi determinantul acesteia.

Funcția are ca parametri matricea sistemului, gradul sistemului, vectorul termenilor liberi, limita sub care pivotul este considerat *0* (pivot 0 înseamnă o coloană nulă, deci matrice neinversabilă şi sistem cu o infinitate de soluții), matricea în care se va scrie inversa matricei sistemului şi vectorul în care se va scrie soluția sistemului. Prin numele funcției se întoarce valoarea determinantului, care are şi rol de parametru de eroare (determinant nul înseamnă matrice neinversabilă).

```
float inversa(float a[][20],int n,float b[],float eps,
              float inv[][20],float x[])
{ float c[20][20],e[20][20],d,aux;
  int i,j,k,p;
  d=1;                              //construire matrice de lucru
  for(i=0;i<n;i++)
   {for(j=0;j<n;j++)
     {c[i][j]=a[i][j]; c[i][j+n]=(i==j);}
    c[i][2*n]=b[i];}
  afisare(c,n,2*n+1);
  i=0;
  while((i<n)&&d)                   //pivotare si calcul determinant
   {p=i;//cautare pivot pe coloana i
    for(j=i+1;j<n;j++)
      if(abs(c[j][i])>abs(c[p][i]))p=j;
    if(abs(c[p][i])<eps)d=0;    //aducere pivot in pozitie
    else{if(p!=i){aux=c[p][i];c[p][i]=c[i][i];c[i][i]=aux;d=-d;}
         d=d*c[i][i];
    for(j=0;j<n;j++) e[j][i]=0; //pivotare
    for(j=i;j<2*n+1;j++) e[i][j]=c[i][j]/c[i][i];
    for(j=0;j<n;j++)
      for(k=i;k<2*n+1;k++)
        if(j!=i)e[j][k]=(c[j][k]*c[p][i]-c[j][i]*c[i][k])/c[p][i];
    for(k=0;k<n;k++)
      for(j=i;j<2*n+1;j++)
        c[k][j]=e[k][j];}
    i++;}
  for(i=0;i<n;i++)              //separare rezultate
   {for(j=0;j<n;j++)
      inv[i][j]=c[i][j+n];
    x[i]=c[i][2*n];}
  return d;}
```

2.5 Lucrul cu pointeri

i. Să se scrie funcţia pentru calculul sumei elementelor unei matrice folosind pointeri pentru adresare.

Funcţia are ca parametri adresa matricei şi dimensiunile ei. Prin numele funcţiei se întoarce suma elementelor matricei. Matricea a cărei adresă a fost primită este memorată în heap. Modificînd primul parametru al funcţiei în *float a[][20]*, se poate folosi funcţia pentru calculul sumei elementelor unei matrice statice cu 20 de coloane.

```
float suma(float **a,int m,int n)
{ int i,j; float s; s=0;
  for(i=0;i<m;i++)
    for(j=0;j<n;j++)
      s+=*(*(a+i)+j);
  return s;}
```

ii. Să se scrie funcția pentru citirea unei matrice de la tastatură şi memorarea ei în heap.

Funcția are ca parametri adresele unde va scrie dimensiunile matricei şi întoarce, prin numele ei, adresa matricei memorate în heap.

```
float ** citire(int *m,int *n)
{ int i,j;
  float **a;
  printf("nr linii: "); scanf("%d", m);
  printf("nr col  : "); scanf("%d", n);
  a=(float **)malloc(*m*sizeof(float*));
  for(i=0;i<*m;i++)
    a[i]=(float*)malloc(*n*sizeof(float));
  for(i=0;i<*m;i++)
    for(j=0;j<*n;j++)
     {printf("a[%d,%d]=",i,j);
      scanf("%f",&a[i][j]);}
  return a;}
```

iii. Să se scrie funcția pentru citirea unui vector şi memorarea lui în heap.

Funcția primeşte ca parametru adresa unde va scrie numărul elementelor vectorului şi returnează, prin numele ei, adresa vectorului memorat în heap.

```
float* citire(int* n)
{ int i;
  float* v;
  v=(float*)malloc(*n*sizeof(float));
  printf("n=");scanf("%d",n);
  for(i=0;i<*n;i++)
   {printf("v(%d)=",i);
    scanf("%f",&v[i]);}
  return v;}
```

iv. Să se scrie funcția pentru sortarea unui vector folosind adresarea cu pointeri.

Funcția are ca parametri adresa vectorului şi numărul său de elemente. Nu se întoarce nici o valoare prin numele funcției. Se poate apela atît pentru vectori memorați în heap, cît şi pentru vectori memorați static.

```
void bule(float *v, int n)
{ int i,p; float a;
  p=1;
  while(p)
   {p=0;
    for(i=0;i<n-1;i++)
      if(*(v+i)>*(v+i+1))
       {a=*(v+i);
        *(v+i)=*(v+i+1);
        *(v+i+1)=a; p=1;}
   }
}
```

2.6 Lucrul cu relaţii (funcţii şi permutări)

Fie A, B două mulţimi de numere reale. O relaţie R este o submulţime a lui $A \times B$; cu alte cuvinte, o relaţie ataşează unui element din A un element din B. Reprezentarea în calculator a unei relaţii se poate face prin intermediul unui tablou bidimensional cu 2 linii şi $|R|$ coloane. O funcţie $f: A \to B$ este o relaţie cu proprietatea că fiecare element al lui A are un corespondent unic în B ($\forall a \in A, \exists! b \in B$ *cu* $f(a) = b$). Dacă mulţimile A şi B sunt finite, atunci o funcţie, ca şi o relaţie, poate fi reprezentată sub forma unui tablou cu 2 dimensiuni.

În acest subcapitol se vor folosi tipurile de date:

MULTIME, definit anterior;

RELATIE, definit astfel:

```
typedef struct{float r[2][100];
               int n;}RELATIE;
```

Cîmpul *r* constituie descrierea relaţiei, iar *n* este numărul de perechi din descrierea relaţiei (cardinalul relaţiei). Tipul RELATIE va fi folosit atît pentru reprezentarea funcţiilor, cît şi a permutărilor.

PERMUTARE, definit astfel:

```
typedef struct{int x[100];
               int n;}PERMUTARE;
```

Cîmpul *x* conţine permutarea, iar cîmpul *n* ordinul permutării.

i. Să se scrie subprogramul care verifică faptul că o relaţie este o funcţie (în sens matematic).

Pentru a verifica dacă o relaţie R este, în particular, o funcţie, va fi suficient să verificăm că $|R| = |A|$ şi că printre elementele primei linii din reprezentarea lui R nu se află dubluri (orice element din prima linie apare o singură dată printre elementele primei linii).

Subprogramul are ca parametri relaţia care trebuie verificată şi mulţimea de definiţie. Prin numele funcţiei se întoarce valoarea 0 dacă relaţia nu este funcţie sau 1 dacă relaţia este funcţie.

```
int este_functie(RELATIE r,MULTIME a)
{ int dublura=0,i,j;
  if(r.n!=a.n) return 0;
  else{for(i=0;(i<r.n-1)&&!dublura;i++)
         for(j=i+1;(j<r.n)&&!dublura;j++)
           if(r.r[0][i]==r.r[0][j])dublura=1;
       return !dublura;}
}
```

ii. Să se scrie subprogramul care verifică dacă o funcție este injectivă.

O funcție *f* este injectivă dacă pe a doua linie a matricei de reprezentare nu există dubluri.

Subprogramul are ca parametru funcția şi întoarce valoarea 0 dacă funcția nu este injectivă sau 1 dacă este injectivă.

```
int este_injectiva(RELATIE r)
{ int injectiva=1,i,j;
  for(i=0;(i<r.n-1)&&injectiva;i++)
    for(j=i+1;(j<r.n)&&injectiva;j++)
      if(r.r[1][i]==r.r[1][j])injectiva=0;
  return injectiva;}
```

iii. Să se scrie subprogramul care verifică dacă o funcție este surjectivă.

În termenii reprezentării anterioare, o funcție *f* este surjectivă dacă a doua linie a matricei de reprezentare conține toate elementele mulțimii B.

Subprogramul are ca parametri funcția şi mulțimea de valori. Prin numele subprogramului se întoarce valoarea 0 dacă relația nu este surjectivă sau 1 dacă este surjectivă.

```
int este_surjectiva(RELATIE r,MULTIME b)
{ int surjectiva=1,gasit,i,j;
  if(n>p) return 0;
  else{for(i=0;(i<b.n)&&surjectiva;i++)
        {for(j=0,gasit=0;(j<b.n)&&!gasit;j++)
           if(r.r[1][j]==b.x[i])gasit=1;
             if(!gasit)surjectiva=0;}
       return surjectiva;}
}
```

iv. Să se scrie subprogramul care compune două funcții.

Fie $f:A\to B$, $g:B\to C$ două funcții. Funcția $h:A\to C$, $h = g \circ f$, $\forall a\in A$, $h(a) = g(f(a))$ se numește compunerea funcției *g* cu funcția *f*. Se va presupune în continuare că cele două funcții, *f* şi *g*, pot fi compuse.

Subprogramul are ca parametri relațiile care reprezintă cele două funcții care se compun şi adresa unde va scrie relația care reprezintă funcția rezultat. Nu se întoarce nici o valoare prin numele funcției.

```
void compunere(RELATIE f,RELATIE g,RELATIE *h)
{ int gasit,i,j;
  for(i=0;i<2;i++)
  for(i=0;i<f.n;i++)
   {h->r[0][i]=f.r[0][i];
    for(j=0,gasit=0;(j<g.n)&&(!gasit);j++)
      if(f.r[1][i]==g.r[0][j])gasit=1;
    h->r[1][i]=g.r[1][--j];}
}
```

v. Să se scrie funcţia care determină dacă o permutare este identică.

Funcţia are ca parametru permutarea şi întoarce valoarea 1 dacă este identică sau 0 în caz contrar.

```
int este_identitatea(PERMUTARE p)
{ int i,estei;
  for(i=0,estei=1;(i<p.n)&&estei;i++)
    if(p.x[i]-i)estei=0;
  return estei;}
```

vi. Să se scrie funcţia care determină dacă o permutare este transpoziţie.

Funcţia are ca parametru permutarea şi întoarce poziţia transpoziţiei (dacă *i* este poziţia întoarsă, atunci transpoziţia este între elementele *i* şi *p.x[i]*) sau -1, dacă permutarea nu este transpoziţie.

```
int este_transpozitie(int *a,int m,int *j)
{ int dif=0,i,j;
  for(i=0;(i<p.n)&&(dif<3);i++)
    if(a[i]!=i)
     {dif++;
      j=i;}
  if(dif!=2) return -1;
  else return j;}
```

vii. Să se scrie funcţia care calculează numărul de permutări ale unei transpoziţii şi (implicit) signatura acesteia.

Funcţia are ca parametru permutarea şi întoarce signatura acesteia.

```
int nr_inversiuni(PERMUTARE p)
{
 if(!p.n) return 0;
 else return(p.x[p.n-1]!=(p.n-1))+nr_inversiuni(p);
}
```

viii. Să se scrie funcţia care compune două permutări de acelaşi ordin.

Funcţia are ca parametri cele două permutări şi adresa unde va scrie noua permutare.

```
void compunere(PERMUTARE a,PERMUTARE b,PERMUTARE *c)
{ int i;
  for(i=0;i<a.n;c->x[i++]=a.x[b.x[i]]);
  c->n=a.n;}
```

ix. Să se scrie funcţia care calculează inversa unei permutări.

Funcţia are ca parametri permutarea şi adresa unde va scrie permutarea inversă.

```
void inversa(PERMUTARE p, PERMUTARE *p1)
{ int gasit,i,k;
  p1->n=p.n;
  for(i=0;i<p.n;i++)
   {for(k=0,gasit=0;(k<p.n)&&(!gasit);k++)
              if(p.x[k]==i)gasit=1;
       p1->x[i]=--k;}
  return p1;}
```

x. Să se scrie funcţia care descompune o permutare în produs de transpoziţii.

Dacă *f* este permutarea a cărei descompunere este dorită şi *k* număr natural astfel încît $f_k=e$, unde $f_1 = f \circ \tau_{i_1 j_1}$, $f_r = f_{r-1} \circ \tau_{i_r j_r}$, $r \geq 2$, unde $\tau_{i_r j_r}$ sînt transpoziţii, atunci $f= \tau_{i_k j_k} \circ \tau_{i_{k-1} j_{k-1}} \circ \cdots \circ \tau_{i_1 j_1}$.

Compunerea permutării f cu transpoziţia τ_{ij} este $g = f \circ \tau_{ij}$, unde

$$g(s) = \begin{cases} f(s), s \neq i, s \neq j \\ f(j), s = i \\ f(i), s = j \end{cases} .$$

Funcţia *descompune(p,n,t,k)* calculează pentru permutarea $p \in S_n$ o valoare k astfel încît $f_k=e$ şi matricea t cu k linii şi două coloane, unde $t[r,1]=i_r$, $t[r,2]=j_r$, $r=1,...,k$. La fiecare pas r, $r=1,...,k$, i este cea mai mică valoare pentru care $f_{r-1}(i) \neq i$, unde $f_0=f$.

Funcţia are ca parametri permutarea şi matricea în care va scrie factorii produsului de transpoziţii. Prin numele funcţiei se întoarce numărul de factori ai produsului de transpoziţii.

```
int transpozitii(PERMUTARE p, int tr[][2])
{int gata=0,aux,j=0,k=0,gasit;
 while(!gata)
  {for(int i=j,gasit=0;(i<p.n)&&!gasit;i++)
     if(p.x[i]-i)
      {gasit=1;j=i;tr[k][0]=i;tr[k][1]=p.x[i];k++;
       aux=a[i];p.x[i]=p.x[aux];p.x[aux]=aux;}
   gata=!gasit;}
  return k;}//nr. de transpozitii
```

Notă: Deoarece nu dispunem de modalităţi de evaluare a ordinului de mărime a valorii *k*, prin utilizarea datelor de tip static este posibilă depăşirea memoriei alocate tabloului *t*. Propunem ca exerciţiu scrierea unei variante utilizînd structuri de date dinamice.

3. Lucrul cu şiruri de caractere

i. Să se scrie funcția care numără cuvintele dintr-un text. Textul se termină cu caracterul punct, iar cuvintele sînt separate prin unul sau mai multe spații.

Funcția are ca parametru şirul de caractere (textul) şi întoarce, prin numele ei, numărul de cuvinte din text.

```
int numara(char s[])
{ int n,i;
  i=0;
  if(s[0]==' ') n=0;
  else n=1;
  while(s[i]!='.')
   {if((s[i]==' ')&&(s[i+1])!=' ') n++;
    i++;}
  if(s[i-1]==' ') n--;
  return(n);}
```

ii. Să se scrie funcția pentru determinarea lungimii celui mai lung cuvînt dintr-un text terminat cu caracterul punct. Cuvintele sînt separate prin unul sau mai multe spații.

Funcția are ca parametru şirul de caractere (textul) şi întoarce, prin numele ei, lungimea celui mai lung cuvînt din text.

```
int maxim(char s[])
{ int i,max,k;
  i=max=k=0;
  while(s[i]!='.')
   {if(s[i]!=' ')k++;
    else {if(k>max) max=k;
          k=0;}
    i++;}
  if((s[i-1]!=' ')&&(k>max))max=k;
  return(max);}
```

iii. Să se scrie funcţia pentru determinarea lungimii celui mai scurt cuvînt dintr-un text terminat cu caracterul punct. Cuvintele sînt separate prin unul sau mai multe spaţii.

Funcţia are ca parametru şirul de caractere (textul) şi întoarce, prin numele ei, lungimea celui mai scurt cuvînt din text.

```
int minim(char s[])
{ int i,min,k;
  i=k=0;min=strlen(s);
  while(s[i]!='.')
   {if(s[i]!=' ') k++;
    else {if((k<min)&&(k!=0)) min=k;
          k=0;}
    i++;}
  if((s[i-1]!=' ')&&(k<min))min=k;
  return(min);}
```

iv. Să se scrie funcţia pentru determinarea frecvenţei de apariţie a fiecărui cuvînt într-un text (şir de caractere). Cuvintele sînt separate prin spaţii.

Funcţia ajutătoare *separa* are rolul de a extrage din text următorul cuvînt, începînd cu poziţia dată. Ea are ca parametri textul în care caută un cuvînt, poziţia de unde începe căutarea şi adresa unde va scrie cuvîntul extras.

```
int separa(char *s, int p, char *c)
{ int i;
  while(s[p]==' ') p++;
  i=p;
  while(s[i]!=' ') {c[i-p]=s[i]; i++;}
  c[i-p]='\0';
  return i;}
```

Funcţia *numarare* are ca parametri şirul de caractere (textul), vectorul de cuvinte (matrice de caractere) în care va scrie cuvintele găsite în text şi vectorul în care va scrie numărul de apariţii al fiecărui cuvînt reţinut în vectorul de cuvinte.

```
int numarare(char *s,char m[][30],int f[])
{ char c[30];
  int i,j,nr,p,k;
  nr=0;p=0;
  while(p<strlen(s))
   {p=separa(s,p,c);
    k=0; i=0;
    while((i<nr)&&(!k))
    if(!strcmp(m[i],c))k=1;
    else i++;
    if(k) f[i]++;
    else {f[nr]=1;
          strcpy(m[nr],c);
          nr++;}
   }
 return nr;}
```

v. Să se scrie funcția pentru sortarea alfabetică a unui vector de şiruri de caractere.

Funcția are ca parametri matricea de caractere în care fiecare linie reprezintă un cuvînt de maxim 100 caractere (vector de cuvinte) şi numărul de cuvinte. Nu se întoarce nici un rezultat prin numele funcției.

```
void sort(char s[][100],int nr)
{ int i,p;
  char t[100];
  p=1;
  while(p)
   {p=0;
    for(i=0;i<nr-1;i++)
      if(strcmp(s[i],s[i+1])>0)
       {p=1;
        strcpy(t,s[i]);
        strcpy(s[i],s[i+1]);
        strcpy(s[i+1],t);}
   }
}
```

vi. Să se scrie funcția care calculează numărul de apariții ale unui caracter într-un şir de caractere.

Funcția are ca parametri şirul de caractere şi caracterul căutat şi întoarce, prin numele ei, numărul de apariții ale acestuia.

```
int frecv_c(char s[], char c)
{ int i,nr;
  nr=0;
  for(i=0;i<strlen(s);i++)
    if(c==s[i]) nr++;
  return nr;}
```

vii. Să se scrie funcția care calculează numărul de apariții ale fiecărui caracter din alfabetul limbii engleze într-un şir de caractere (nu se ține cont de litere mari sau mici).

Funcția are ca parametri şirul de caractere (textul) şi vectorul în care va scrie frecvențele fiecărui caracter ASCII din alfabetul englez, în ordinea codurilor acestora (‘A’-‘Z’).

```
void frecventa(char s[], int f[])
{ int i;
  for(i=0;i<256;i++) f[i]=0;
  for(i=0;i<strlen(s);i++)
    if((toupper(s[i])>='A')&&(toupper(s[i])<='Z'))
      f[toupper(s[i])-'A']++;}
```

viii. Să se scrie funcţia care calculează numărul de apariţii ale fiecărui caracter ASCII într-un şir de caractere.

Funcţia are ca parametri şirul de caractere (textul) şi vectorul în care va scrie frecvenţele fiecărui caracter ASCII, în ordinea codurilor acestora.

```
void frecv(char s[], int f[])
{ int i;
  for(i=0;i<256;i++) f[i]=0;
  for(i=0;i<strlen(s);i++) f[s[i]]++;}
```

ix. Să se scrie funcţia pentru criptarea şi decriptarea unui şir de caractere prin interschimbarea grupurilor de 4 biţi din reprezentarea fiecărui caracter.

Funcţia are ca parametru adresa şirului pe care trebuie să îl (de)cripteze. Ea realizează atît operaţia de criptare, cît şi cea de decriptare, fiind propria ei inversă.

```
void cripteaza(char* s)
{ int i;
  unsigned char m1=15,m2=240,a,b;
  for(i=0;i<strlen(s);i++)
    {a=(s[i]&m1)<<4; b=(s[i]&m2)>>4; s[i]=a|b;}}
```

x. Să se scrie funcţiile pentru criptarea şi decriptarea unui şir de caractere prin rotirea setului de caractere cu 13 poziţii (algoritmul ROT13).

Ambele funcţii au ca parametru adresa şirului care trebuie (de)criptat.

```
void cripteaza(char* s)
{ int i;
  for(i=0;i<strlen(s);i++) s[i]=(s[i]+13)%256;}
void decripteaza(char* s)
{ int i;
  for(i=0;i<strlen(s);i++) s[i]=(s[i]+243)%256;}
```

xi. Să se scrie funcţia pentru găsirea poziţiei de început a unui subşir într-un şir de caractere.

Funcţia are ca parametri adresa şirului în care se caută şi adresa subşirului căutat. Prin numele funcţiei se întoarce poziţia subşirului în şir, dacă este găsit, sau -1, dacă nu este găsit.

```
int cauta(char *s, char *ss)
{ int i,j,k;
  k=-1; i=0;
  while((i<strlen(s))&&(k==-1))
    if(s[i]!=ss[0]) i++;
    else {j=1; k=i;
          while(j<strlen(ss)&&(k!=-1))
            if(ss[j]!=s[i+j])k=-1;
            else j++;
          i++;}
  return k;}
```

4. Structuri dinamice de date

4.1 Liste simplu înlănțuite

În exercițiile din acest capitol se va folosi o listă avînd ca informație utilă în nod un număr real:

```
typedef struct TNOD{float x;
                    struct TNOD* next;};
```

Nu se vor verifica posibilele erori la rezervarea spațiului în heap, cauzate de lipsa acestuia. Pentru verificarea acestui tip de eroare se poate testa rezultatul întors de funcția *malloc* (NULL dacă nu s-a putut face alocarea de memorie).

Pentru uşurință în scriere, se va folosi următoarea macrodefiniție:

```
#define NEW (TNOD*)malloc(sizeof(TNOD));
```

i. Să se scrie programul pentru crearea unei liste simplu înlănțuite cu preluarea datelor de la tastatură. Sfîrşitul introducerii datelor este marcat standard. După creare, se va afişa conținutul listei apoi se va elibera memoria ocupată.

```
#include<stdio.h>
#include<alloc.h>
typedef struct TNOD{float x;
                    struct TNOD* next;};
void main()
{TNOD *cap,*p,*q;
 int n,i;
 float a;
 //creare lista
 printf("primul nod="); scanf("%f",&a);
 cap=(TNOD*)malloc(sizeof(TNOD));
 cap->x=a; cap->next=NULL;
 p=cap;
```

```
 printf("nod (CTRL-Z)=");
 scanf("%f",&a);
 while(!feof(stdin))
 {q=(TNOD*)malloc(sizeof(TNOD));
  q->x=a; q->next=NULL;
  p->next=q;
  p=q;
  printf("nod (CTRL-Z)=");
  scanf("%f",&a);}
 //afisare continut
 printf("\n");
 p=cap;
 while(p)
 {printf("\t%5.2f",p->x);
  p=p->next;}
 //stergere lista
 while(cap)
 {p=cap; cap=cap->next;
  free(p);}
}
```

ii. Să se scrie funcția pentru inserarea unui nod la începutul unei liste simplu înlănțuite.

Funcția are ca parametri capul listei în care se inserează şi valoarea care se inserează. Prin numele funcției se întoarce noul cap al listei.

```
TNOD* ins1(TNOD* cap, float a)
{TNOD* p;
 p=NEW;
 p->x=a;
 p->next=cap;
 return p;
}
```

iii. Să se scrie funcția pentru inserarea unui nod într-o listă simplu înlănțuită după un nod identificat prin valoarea unui cîmp. Dacă nodul căutat nu există, inserarea se face la sfîrşitul listei.

Funcția are ca parametri capul listei în care se inserează, valoarea care se inserează şi valoarea după care se inserează. Prin numele funcției se întoarce noul cap al listei.

```
TNOD* ins2(TNOD* cap, float a,float x)
{TNOD *p,*q;
 p=NEW;
 p->x=a;
 if(!cap) {cap=p;p->next=NULL;}
 else{q=cap;
      while((q->next)&&(q->x!=x))
        q=q->next;
      p->next=q->next;
      q->next=p;}
 return cap;}
```

iv. Să se scrie funcția pentru inserarea unui nod într-o listă simplu înlănțuită înaintea unui nod identificat prin valoarea unui cîmp. Dacă nodul căutat nu există, se face inserare la începutul listei.

Funcția are ca parametri capul listei în care se inserează, valoarea care se inserează şi valoarea înaintea cărei se inserează. Prin numele funcției se întoarce noul cap al listei.

```
TNOD* ins3(TNOD* cap, float a,float x)
{TNOD *p,*q;
 p=NEW; p->x=a;
 if(!cap) {cap=p;p->next=NULL;}
 else{q=cap;
      while((q->next)&&(q->next->x!=x))
        q=q->next;
      if(!q->next){p->next=cap;cap=p;}
      else {p->next=q->next;
            q->next=p;}
      }
 return cap;}
```

v. Să se scrie funcția pentru căutarea unui nod identificat prin valoarea unui cîmp într-o listă simplu înlănțuită.

Funcția are ca parametri capul listei în care se caută şi valoarea căutată. Prin numele funcției se întoarce adresa nodului care conține informația căutată sau NULL, dacă informația nu a fost găsită în listă. Dacă sînt mai multe noduri cu aceeaşi valoare, se întoarce adresa ultimului.

```
TNOD* cauta(TNOD* cap, float a)
{TNOD *q;
 q=cap;
 while((q)&&(q->x!=a))
   q=q->next;
 return q;}
```

vi. Să se scrie funcția pentru adăugarea unui nod la sfîrşitul unei liste simplu înlănțuite.

Funcția are ca parametri capul listei în care se inserează şi valoarea care se inserează. Prin numele funcției se întoarce noul cap al listei.

```
TNOD* ins4(TNOD* cap, float a)
{TNOD *p,*q;
 p=NEW;
 p->x=a;
 p->next=NULL;
 if(!cap) cap=p;
 else{q=cap;
      while(q->next)
        q=q->next;
      q->next=p;}
 return cap;}
```

vii. Să se scrie funcţia pentru ştergerea primului nod al unei liste simplu înlănţuite.

Funcţia are ca parametru capul listei din care se şterge primul nod şi întoarce, prin numele ei, noul cap al listei.

```
TNOD* sterg1(TNOD* cap)
{TNOD *p;
 if(cap){p=cap; cap=cap->next;
         free(p);}
 return cap;}
```

viii. Să se scrie funcţia pentru ştergerea unui nod identificat prin valoarea unui cîmp dintr-o listă simplu înlănţuită.

Funcţia primeşte ca parametri adresa capului listei şi valoarea informaţiei utile din nodul care se şterge. Funcţia întoarce valoarea 1, dacă nu a găsit nodul căutat, sau 0, dacă a efectuat ştergerea.

```
int sterg2(TNOD** cap,float a)
{TNOD *p, *q;
 int er;
 if(*cap){p=*cap;
          if((*cap)->x==a){*cap=(*cap)->next;
                           er=0;
                           free(p);}
          else {while((p->next)&&(p->next->x!=a))
                  p=p->next;
                if(p->next){q=p->next;
                            p->next=p->next->next;
                            free(q);
                            er=0;}
                else er=1;}
         }
 else er=1;
 return er;}
```

ix. Să se scrie funcţia pentru ştergerea ultimului nod dintr-o listă simplu înlănţuită.

Funcţia are ca parametru capul listei din care se şterge ultimul nod şi întoarce, prin numele ei, noul cap al listei.

```
TNOD* sterg3(TNOD* cap)
{TNOD *p;
 p=cap;
 if(cap)
  if(!cap->next){free(cap);
                 cap=NULL;}
  else{while((p->next->next))
         p=p->next;
       free(p->next);
       p->next=NULL;}
 return cap;}
```

x. Să se scrie funcția pentru ştergerea unei liste simplu înlănțuite.

Funcția are ca parametru capul listei care trebuie ştearsă şi întoarce, prin numele ei, noul cap al listei (valoarea NULL).

```
TNOD* sterg(TNOD* cap)
{TNOD* p;
 while(cap)
  {p=cap;
   cap=cap->next;
   free(p);}
 return NULL;}
```

xi. Să se scrie funcția pentru ştergerea nodului aflat după un nod identificat prin valoarea unui cîmp.

Funcția are ca parametri capul listei şi valoarea nodului căutat (după care se şterge). Prin numele funcției se întoarce valoarea 0, dacă s-a făcut ştergerea, sau 1, dacă nodul căutat nu a fost găsit.

```
int sterg4(TNOD* cap,float a)
{TNOD *p, *q;
 int er;
 if(!cap) er=1;
 else{p=cap;
      while((p)&&(p->x!=a))
           p=p->next;
      if(!p)er=1;
      else{q=p->next;
           if(q){p->next=q->next;
                 er=0;
                 free(q);}
           else er=1;}
     }
 return er;}
```

xii. Să se scrie funcția pentru ştergerea nodului aflat înaintea nodului identificat prin valoarea unui cîmp.

Funția are ca parametri capul listei, valoarea din nodul căutat (înaintea căruia se face ştergerea) şi adresa unde se înscrie parametrul de eroare (0 dacă se face ştergerea, 1 dacă nodul căutat este primul, 2 dacă nodul căutat nu a fost găsit). Funcția întoarce noul cap al listei.

```
TNOD* sterg5(TNOD* cap,float a, int *er)
{TNOD *p, *q;
 if(!cap) *er=2;
 else{p=cap;
      if(cap->x==a)*er=1;
      else if(cap->next)
             if(cap->next->x==a)
               {*er=0;
                p=cap; cap=cap->next;
                free(p);}
```

```
           else if(cap->next->next)
                  {p=cap;
                   while((p->next->next)&&(p->next->next->x!=a))
                     p=p->next;
                   if(p->next->next)
                     {q=p->next;
                      p->next=p->next->next;
                      free(q);
                    * er=0;}
                   else *er=2;
                  }
     }
 return cap;}
```

4.2 Stive şi cozi

În exerciţiile acestui subcapitol se va folosi o stivă de elemente cu tip real. Pentru implementarea ei se va folosi o listă simplu înlănţuită. Capul acestei liste, împreună cu capacitatea maximă a stivei şi numărul de poziţii ocupate constituie cîmpuri ale structurii STIVA:

```
typedef struct TNOD{float x;
                    struct TNOD* next;};
typedef struct STIVA{TNOD* vf;
                     capacitate c,max;};
```

i. Să se scrie funcţia care verifică dacă o stivă este vidă.

Funcţia are ca parametru stiva şi întoarce valoarea 1 dacă stiva este vidă sau 0 în caz contrar.

```
int e_vida(STIVA s)
{return s.vf==NULL;}
```

ii. Să se scrie funcţia care verifică dacă o stivă este plină.

Funcţia are ca parametru stiva şi întoarce valoarea 1, dacă stiva este plină, sau 0, în caz contrar.

```
int e_plina(STIVA s)
{return s.c==s.max;}
```

iii. Să se scrie funcţia pentru adăugarea unui element de tip real într-o stivă.

Funcţia are ca parametri adresa stivei şi valoarea care trebuie adăugată în stivă. Prin numele funcţiei se întoarce valoarea 1, dacă stiva era plină (şi nu se poate face adăugare), sau 0, dacă adăugarea a decurs normal.

```
int push(STIVA *s, float a)
{TNOD *p;
 int er;
 if(!e_plina(*s))
 {p=NEW;   p->x=a;
  p->next=s->vf;
  s->vf=p; (s->c)++;
  er=0;}
 else er=1;
 return er;}
```

iv. Să se scrie funcția pentru extragerea unui element dintr-o stivă.

Funcția are ca parametri adresa stivei şi adresa unde se va depune valoarea extrasă din stivă. Prin numele funcției se întoarce valoarea 1, dacă stiva este vidă, sau 0, în caz contrar.

```
int pop(STIVA *s, float *a)
{int er;
 TNOD *p;
 if(e_vida(*s))er=1;
 else{p=s->vf;
      s->vf=s->vf->next;
      *a=p->x;
      free(p);
      er=0; (s->c)--;}
 return er;}
```

v. Să se scrie funcția pentru golirea unei stive.

Funcția are ca parametru adresa stivei şi nu întoarce nimic, prin numele ei. Valorile aflate în stivă se pierd.

```
void golire(STIVA *s)
{TNOD *p;
 while(s->vf)
 {p=s->vf;
  s->vf=s->vf->next;
  free(p);}
 s->c=0;}
```

Programul următor exemplifică modul de lucru cu funcțiile descrise mai sus:

```
#include<stdio.h>
#include<alloc.h>
#define NEW (TNOD*)malloc(sizeof(TNOD));
#define capacitate int

typedef struct TNOD{float x;
                    struct TNOD* next;};
typedef struct STIVA{TNOD* vf;
                     capacitate c,max;};
```

```
int e_plina(STIVA s)
{return s.c==s.max;}

int e_vida(STIVA s)
{return s.vf==NULL;}

void afisare(STIVA s)
{if(e_vida(s))printf("\nStiva vida.");
 else{printf("\n");
      while(s.vf)
        {printf("\t%5.2f",s.vf->x);
         s.vf=s.vf->next;}
     }
}

void golire(STIVA *s)
{TNOD *p;
 while(s->vf)
 {p=s->vf;
  s->vf=s->vf->next;
  free(p);}
 s->c=0;}

int push(STIVA *s, float a)
{TNOD *p;
 int er;
 if(!e_plina(*s))
 {p=NEW;   p->x=a;
  p->next=s->vf;
  s->vf=p; (s->c)++;
  er=0;}
 else er=1;
 return er;}

int pop(STIVA *s, float *a)
{int er;
 TNOD *p;
 if(e_vida(*s))er=1;
 else{p=s->vf;
      s->vf=s->vf->next;
      *a=p->x;
      free(p);
      er=0; (s->c)--;}
 return er;}

void main()
{STIVA s;
 int n;
 float a;
//initializare stiva
 s.vf=NULL;s.c=0;
 afisare(s);
 do {printf("\ncap max=");scanf("%d",&s.max);}
 while(s.max<=0);
 printf("\ninfo:");scanf("%f",&a);
```

```
 while(!feof(stdin))
 {if(push(&s,a)==1)printf("\nStiva plina!");
  afisare(s);
  printf("\ninfo:");scanf("%f",&a);}
//extragere element
 n=pop(&s,&a);
 printf("\n rezultat: %d, %5.3f",n,a);
 afisare(s);
//golire stiva
 printf("\n golire: \n");
 while(!pop(&s,&a))
  printf("\t%5.2f",a);
 afisare(s);
}
```

În exercițiile care urmează se vor folosi cozi de elemente de tip real. Pentru implementarea ei se va folosi o listă simplu înlănțuită (ca în exercițiile precedente). Capul listei, împreună cu capacitatea maximă a cozii, numărul de poziții ocupate şi coada cozii (adresa ultimului element) constituie cîmpuri ale structurii COADA:

```
typedef struct TNOD{float x;
                    struct TNOD* next;};
typedef struct COADA{TNOD *cap, *coada;
                     capacitate c,max;};
```

vi. Să se scrie funcția care verifică dacă o coadă este vidă.

Funcția are ca parametru coada şi întoarce valoarea 1 dacă aceasta este vidă sau 0 în caz contrar.

```
int e_vida(COADA c)
{return c.cap==NULL;}
```

vii. Să se scrie funcția care verifică dacă o coadă este plină.

Funcția primeşte ca parametru coada şi întoarce valoarea 1, dacă aceasta este plină, sau 0, în caz contrar.

```
int e_plina(COADA c)
{return c.c==c.max;}
```

viii. Să se scrie funcția pentru adăugarea unui element de tip real într-o coadă.

Funcția are ca parametri adresa cozii şi valoarea care trebuie adăugată. Prin numele funcției se întoarce valoarea 1, cînd coada este plină (şi nu se poate face adăugare), sau 0, dacă adăugarea a decurs normal.

```
int push_c(COADA *c, float a)
{TNOD *p;
 int er;
 if(!e_plina(*c))
 {p=NEW;
```

```
  p->x=a;
  p->next=NULL;
  if(c->coada)  c->coada->next=p;
  else c->cap=p;
  c->coada=p;
  (c->c)++;
  er=0;}
 else er=1;
 return er;}
```

ix. Să se scrie funcţia pentru extragerea unui element dintr-o coadă.

Funcţia are ca parametri adresa cozii şi adresa unde se va depune valoarea extrasă. Prin numele funcţiei se întoarce valoarea 1, cînd coada este vidă, sau 0, în caz contrar.

```
int pop_c(COADA *c, float *a)
{int er;
 TNOD *p;
 if(e_vida(*c))er=1;
 else{p=c->cap;
      c->cap=c->cap->next;
      *a=p->x;
      free(p);er=0;
      if(--(c->c))c->coada=NULL; }
 return er;}
```

x. Să se scrie funcţia pentru golirea unei cozi.

Funcţia are ca parametru adresa cozii şi nu întoarce nimic, prin numele ei. Valorile aflate în coadă se pierd.

```
void golire(COADA *c)
{TNOD *p;
 while(c->cap)
 {p=c->cap;
  c->cap=c->cap->next;
  free(p);}
  c->c=0;
  c->coada=NULL;}
```

4.3 Liste circulare simplu înlănţuite

i. Să se scrie programul pentru crearea unei liste circulare simplu înlănţuite cu preluarea datelor de la tastatură. Sfîrşitul introducerii datelor este marcat standard. După creare, se va afişa conţinutul listei apoi se va elibera memoria ocupată.

```
#include<stdio.h>
#include<alloc.h>
typedef struct TNOD{float x;
                    struct TNOD* next;};
void main()
{TNOD *cap,*p,*q;
 int n,i;
 float a;
 //creare lista
 printf("primul nod=");
 scanf("%f",&a);
 cap=(TNOD*)malloc(sizeof(TNOD));
 cap->x=a;
 cap->next=cap;
 p=cap;
 printf("nod (CTRL-Z)=");
 scanf("%f",&a);
 while(!feof(stdin))
 {q=(TNOD*)malloc(sizeof(TNOD));
  q->x=a;
  q->next=cap;
  p->next=q;
  p=q;
  printf("nod (CTRL-Z)=");
  scanf("%f",&a);}
 //afisare continut, exista cel putin un nod
 printf("\n");
 printf("\t%5.2f",cap->x);
 p=cap->next;
 while(p!=cap)
 {printf("\t%5.2f",p->x);
  p=p->next;}
 //stergere lista, exista cel putin un nod
 //pe parcursul stergerii lista nu isi pastreaza proprietatea de
 //circularitate
 p=cap;
 while(cap!=p)
 {q=cap;
  cap=cap->next;
  free(q);}
 cap=NULL;}
```

ii. Să se scrie funcția pentru inserarea unui nod la începutul unei liste circulare simplu înlănțuite.

Funcția are ca parametri capul listei şi valoarea care trebuie inserată. Prin numele funcției se returnează noul cap al listei.

```
TNOD* ins1(TNOD* cap, float a)
{TNOD *p,*q;
 p=NEW;
 p->x=a;
 p->next=cap;
 if(!cap)p->next=p;
 else {q=cap;
```

```
       while(q->next!=cap)
         q=q->next;
       q->next=p;}
 return p;}
```

iii. Să se scrie funcţia pentru inserarea unui nod într-o listă circulară simplu înlănţuită după un nod identificat prin valoarea unui cîmp.

Funcţia are ca parametri capul listei, valoarea care trebuie inserată şi valoarea după care se inserează. Dacă valoarea căutată nu este găsită, atunci inserarea se face la sfîrşit.

```
TNOD* ins2(TNOD* cap, float a,float x)
{TNOD *p,*q;
 p=NEW;
 p->x=a;
 if(!cap) {cap=p;p->next=cap;}
 else{q=cap;
      while((q->next!=cap)&&(q->x!=x))
        q=q->next;
      p->next=q->next;
      q->next=p;}
 return cap;}
```

iv. Să se scrie funcţia pentru inserarea unui nod într-o listă circulară simplu înlănţuită înaintea unui nod identificat prin valoarea unui cîmp.

Funcţia are ca parametri capul listei, valoarea care se adaugă şi valoarea înaintea căreia se adaugă. Prin numele funcţiei se întoarce noul cap al listei.

```
TNOD* ins3(TNOD* cap, float a,float x)
{TNOD *p,*q;
 p=NEW;
 p->x=a;
 if(!cap) {cap=p;p->next=cap;}
 else{q=cap;
      while((q->next!=cap)&&(q->next->x!=x))
        q=q->next;
      if(q->next==cap)cap=p;
      p->next=q->next; q->next=p;}
 return cap;}
```

v. Să se scrie funcţia pentru căutarea unui nod, identificat prin valoarea unui cîmp, într-o listă circulară simplu înlănţuită.

Funcţia are ca parametri capul listei şi valoarea căutată. Prin numele funcţiei se întoarce adresa nodului care conţine valoarea căutată (dacă sînt mai multe noduri cu aceeaşi valoare se întoarce adresa ultimului). Dacă nodul căutat nu este găsit, se întoarce valoarea NULL.

```
TNOD* cauta(TNOD* cap, float a)
{TNOD *q;
 q=cap;
 if(cap)
```

```
  {while((q->next!=cap)&&(q->x!=a))
       q=q->next;
   if(q->x!=a)q=NULL;}
 else q=NULL;
 return q;}
```

vi. Să se scrie funcţia pentru adăugarea unui nod la sfîrşitul unei liste circulare simplu înlănţuite.

Funcţia are ca parametri capul listei şi valoarea care trebuie adăugată. Prin numele funcţiei se întoarce noul cap al listei.

```
TNOD* ins4(TNOD* cap, float a)
{TNOD *p,*q;
 p=NEW;
 p->x=a;
 if(!cap) cap=p;
 else{q=cap;
      while(q->next!=cap)
        q=q->next;
      q->next=p;}
 p->next=cap;
 return cap;}
```

vii. Să se scrie funcţia pentru ştergerea primului nod al unei liste circulare simplu înlănţuite.

Funcţia are ca parametru capul listei şi întoarce noul cap al listei.

```
TNOD* sterg1(TNOD *cap)
{TNOD *p, *q;
 if(cap)
  if(cap==cap->next)
    {free(cap);
     cap=NULL;}
  else{q=cap;
       while(q->next!=cap)
         q=q->next;
       p=cap; cap=cap->next;
       q->next=cap;
       free(p);}
 return cap;}
```

viii. Să se scrie funcţia pentru ştergerea unui nod identificat prin valoarea unui cîmp dintr-o listă circulară simplu înlănţuită.

Funcţia are ca parametri adresa capului listei şi valoarea care trebuie ştearsă. Prin numele funcţiei se întoarce codul de eroare, cu semnificaţia: 1 – nu a fost găsită valoarea căutată, 0 – a fost ştearsă valoarea căutată. Pentru ştergerea primului nod al listei se apelează funcţia *sterg1* (exerciţiul 4.3.vii).

```
int sterg2(TNOD** cap,float a)
{TNOD *p, *q;
 int er;
 if(*cap){p=*cap;
          if((*cap)->x==a){er=0;
                           *cap=sterg1(*cap);}
          else {while((p->next!=*cap)&&(p->next->x!=a))
                  p=p->next;
                if(p->next!=*cap){q=p->next;
                                  p->next=p->next->next;
                                  free(q);
                                  er=0;}
                else er=1;}
         }
 else er=1;
 return er;}
```

ix. Să se scrie funcția pentru ştergerea ultimului nod dintr-o listă circulară simplu înlănțuită.

Funcția are ca parametru capul listei şi întoarce, prin numele ei, noul cap al listei.

```
TNOD* sterg3(TNOD* cap)
{TNOD *p,*q;
 p=cap;
 if(cap)
  if(cap==cap->next){free(cap);
                     cap=NULL;}
  else{while((p->next->next!=cap))
         p=p->next;
       q=p->next;
       p->next=cap;
       free(q);}
 return cap;}
```

x. Să se scrie funcția pentru ştergerea unei liste circulare simplu înlănțuite.

Funcția are ca parametru capul listei şi întoarce, prin numele ei, noul cap al listei (NULL).

```
TNOD* sterg(TNOD* cap)
{TNOD *p,*q;
 if(cap)
 {p=cap;

  do
    {q=cap;
     cap=cap->next;
     free(q);}
  while(cap!=p);}
 return NULL;}
```

xi. Să se scrie funcția pentru ştergerea nodului aflat după un nod identificat prin valoarea unui cîmp. Dacă nodul căutat este ultimul, atunci se şterge primul nod al listei.

Funcția are ca parametri capul listei, valoarea căutată şi adresa unde va scrie parametrul de eroare (1, dacă valoarea căutată nu este găsită sau lista are numai un nod, 0, dacă se face ştergerea).

```
TNOD* sterg4(TNOD* cap,float a,int *er)
{TNOD *p, *q;
 if(!cap) *er=1;
 else if(cap==cap->next)*er=1;
      else{p=cap;
           if(cap->x!=a)
             {p=cap->next;
              while((p!=cap)&&(p->x!=a))
                p=p->next;}
              if(p->x!=a)*er=1;
              else {if(p->next==cap)cap=sterg1(cap);
                    else{q=p->next;
                         p->next=q->next;
                         free(q);}
                         *er=0;}
                   }
 return cap;}
```

xii. Să se scrie funcția pentru ştergerea nodului aflat înaintea nodului identificat prin valoarea unui cîmp. Dacă nodul căutat este primul, se va şterge ultimul nod al listei.

Funcția are ca parametri capul listei, valoarea căutată şi adresa unde se va înscrie parametrul de eroare (0 dacă ştergerea s-a efectuat, 1 dacă nodul căutat nu este găsit sau lista are un singur nod, 2 dacă lista este vidă).

```
TNOD* sterg5(TNOD* cap,float a, int *er)
{TNOD *p, *q;
 if(!cap) *er=2;
 else if(cap==cap->next)*er=1;
      else if(cap->x==a){*er=0;
                         cap=sterg3(cap);}
           else if(cap->next->x==a){cap=sterg1(cap);*er=0;}
                else{p=cap->next;
                     while((p->next->next!=cap)&&(p->next->next-
>x!=a))
                        p=p->next;
                     if(p->next->next->x==a)
                       {q=p->next;
                        p->next=p->next->next;
                        free(q);
                        *er=0;}
                      else *er=1;}
 return cap;}
```

4.4 Liste dublu înlănţuite

Pentru exemplificarea lucrului cu liste dublu înlănţuite se va folosi o listă avînd ca informaţie utilă în nod un număr real:

```
typedef struct TNOD{float x;
                    struct TNOD *next,*pred;};
```

i. Să se scrie programul pentru crearea unei liste dublu înlănţuite cu preluarea datelor de la tastatură. Sfîrşitul introducerii datelor este marcat standard. După creare, se va afişa conţinutul listei, apoi se va elibera memoria ocupată.

```
#include<stdio.h>
#include<alloc.h>
#define NEW (TNOD*)malloc(sizeof(TNOD));
typedef struct TNOD{float x;
                    struct TNOD *next,*pred;};
void main()
{TNOD *cap, *p, *q;
 float x;
 //creare lista
 printf("\nprimul nod: ");scanf("%f",&x);
 cap=NEW;
 cap->pred=NULL; cap->next=NULL;
 cap->x=x;
 p=cap;
 printf("\nnod: ");scanf("%f",&x);
 while(!feof(stdin))
 {q=NEW;
  q->next=NULL; q->x=x;
  q->pred=p; p->next=q;
  p=q;
  printf("\nnod: ");scanf("%f",&x);}
 //afisare lista
 printf("\n");
 p=cap;
 while(p)
  {printf("\t%5.2f",p->x);
   p=p->next;}
 //stergere lista
 while(cap)
  {p=cap;
   cap=cap->next;
   if(cap)cap->pred=NULL;
   free(p);}
 printf("\ngata");}
```

ii. Să se scrie funcția pentru inserarea unui nod la începutul unei liste dublu înlănțuite.

Funcția are ca parametri capul listei şi valoarea care trebuie inserată. Prin numele funcției se întoarce noul cap al listei.

```
TNOD* ins1(TNOD* cap, float a)
{TNOD *p;
 p=NEW;
 p->x=a;
 p->pred=NULL;
 p->next=cap;
 if(cap)cap->pred=p;
 return p;}
```

iii. Să se scrie funcția pentru inserarea unui nod într-o listă dublu înlănțuită, după un nod identificat prin valoarea unui cîmp.

Funcția are ca parametri capul listei, valoarea care se inserează şi valoarea după care se inserează. Dacă valoarea căutată nu este găsită, se face inserare la sfîrşitul listei.

```
TNOD* ins2(TNOD* cap, float a, float x)
{TNOD *p,*q;
 p=NEW;
 p->x=a;
 if(!cap){cap=p;p->next=NULL;p->pred=NULL;}
 else{q=cap;
      while((q->next)&&(q->x!=x))
        q=q->next;
      p->next=q->next;
      p->pred=q;
      q->next=p;
      if(p->next)p->next->pred=p;}
 return cap;}
```

iv. Să se scrie funcția pentru inserarea unui nod într-o listă dublu înlănțuită, înaintea unui nod identificat prin valoarea unui cîmp.

Funcția are ca parametri capul listei, valoarea care trebuie inserată şi valoarea înaintea căreia se inserează. Dacă valoarea căutată nu este găsită, se face inserare la începutul listei.

```
TNOD* ins3(TNOD *cap, float a, float x)
{TNOD *p, *q;
 p=NEW;
 p->x=a;
 if(!cap){cap=p;p->next=NULL;p->pred=NULL;}
 else{q=cap;
      while((q->next)&&(q->x!=x))
        q=q->next;
      if(q->x!=x){p->next=cap; cap->pred=p;
                  p->pred=NULL;
                  cap=p;}
```

```
      else {p->next=q;
            p->pred=q->pred;
            q->pred=p;
            if(p->pred)p->pred->next=p;
              else cap=p;}
           }
 return cap;}
```

v. Să se scrie funcţia pentru căutarea unui nod identificat prin valoarea unui cîmp într-o listă dublu înlănţuită.

Funcţia are ca parametri capul listei în care se caută şi valoarea căutată. Prin numele funcţiei se întoarce adresa nodului care conţine informaţia căutată sau NULL dacă informaţia nu a fost găsită în listă. Dacă sînt mai multe noduri cu aceeaşi valoare se întoarce adresa ultimului. Funcţia este identică pentru liste simplu şi dublu înlănţuite.

```
TNOD* cauta(TNOD* cap, float a)
{TNOD *q;
 q=cap;
 while((q)&&(q->x!=a))
   q=q->next;
 return q;}
```

vi. Să se scrie funcţia pentru adăugarea unui nod la sfîrşitul unei liste dublu înlănţuite.

Funcţia are ca parametri capul listei şi valoarea care trebuie inserată. Funcţia întoarce, prin numele ei, noul cap al listei.

```
TNOD* ins4(TNOD* cap, float a)
{TNOD *p, *q;
 p=NEW;
 p->x=a;
 p->next=NULL;
 if(!cap){p->pred=NULL; cap=p;}
 else {q=cap;
       while(q->next)
         q=q->next;
       p->pred=q;
       q->next=p;}
 return cap;}
```

vii. Să se scrie funcţia pentru ştergerea primului nod al unei liste dublu înlănţuite.

Funcţia are ca parametru capul listei şi întoarce, prin numele ei, noul cap al listei.

```
TNOD* sterg1(TNOD* cap)
{TNOD* p;
 if(cap)
 {p=cap;
```

```
  cap=cap->next;
  if(cap)cap->pred=NULL;
  free(p);}
 return cap;}
```

viii. Să se scrie funcția pentru ştergerea unui nod identificat prin valoarea unui cîmp dintr-o listă dublu înlănțuită.

Funcția are ca parametri capul listei şi valoarea de identificare a nodului care trebuie şters. Prin numele funcției se întoarce noul cap al listei.

```
TNOD* sterg2(TNOD* cap, float a)
{TNOD *p, *q;
 if(cap)
  {p=cap;
   while((p->next)&&(p->x!=a))
      p=p->next;
   if(p->x==a)
     {if(p->next)p->next->pred=p->pred;
      if(p->pred)p->pred->next=p->next;
      else cap=p->next;
      free(p);}
  }
 return cap;}
```

ix. Să se scrie funcția pentru ştergerea ultimului nod dintr-o listă dublu înlănțuită.

Funcția are ca parametru capul listei şi întoarce, prin numele ei, noul cap al listei.

```
TNOD* sterg3(TNOD* cap)
{TNOD *p,*q;
 if(cap)
   {p=cap;
    while(p->next)
       p=p->next;
    if(!p->pred){free(cap);cap=NULL;}
    else {q=p->pred;free(p);q->next=NULL;}
   }
 return cap;}
```

x. Să se scrie funcția pentru ştergerea unei liste dublu înlănțuite.

Funcția are ca parametru capul listei şi întoarce, prin numele ei, noul cap al listei (valoarea NULL).

```
TNOD* sterg(TNOD* cap)
{TNOD*p;
 while(cap)
  {p=cap; cap=cap->next;
   if(cap)cap->pred=NULL;
   free(p);}
 return NULL;}
```

xi. Să se scrie funcția pentru ştergerea nodului aflat după un nod identificat prin valoarea unui cîmp.

Funcția are ca parametri capul listei şi valoarea căutată. Prin numele funcției se întoarce parametrul de eroare, cu semnificația: 0 – s-a efectuat ştergerea, 1 – nodul căutat nu a fost găsit.

```
int sterg4(TNOD* cap, float a)
{TNOD *p, *q;
 int er;
 er=1;
 if(cap)
  {p=cap;
   while((p->next)&&(p->x!=a))
      p=p->next;
   if(p->x==a)
     if(p->next)
       {er=0;
        q=p->next;
        p->next=p->next->next;
        if(p->next)p->next->pred=p;
        free(q);}
  }
 return er;}
```

xii. Să se scrie funcția pentru ştergerea nodului aflat înaintea nodului identificat prin valoarea unui cîmp.

Funcția are ca parametri adresa capului listei şi valoarea căutată. Prin numele funcției se întoarce parametrul de eroare, cu semnificația: 0 – s-a efectuat ştergerea, 1 – valoarea căutată nu a fost găsită.

```
int sterg5(TNOD** cap, float a)
{TNOD *p, *q;
 int er;
 er=1;
 if(*cap)
  {p=*cap;
   while((p->next)&&(p->x!=a))
      p=p->next;
   if(p->x==a)
     if(p->pred)
       {er=0;
        q=p->pred;
        p->pred=p->pred->pred;
        if(p->pred)p->pred->next=p;
        else *cap=p;
        free(q);}
  }
 return er;}
```

4.5 Alte exerciții

i. Să se scrie funcția recursivă pentru numărarea nodurilor unei liste simplu înlănțuite.

Funcția are ca parametru capul listei şi întoarce, prin numele ei, numărul de noduri ale listei.

```
int numara_r(TNOD* cap)
{int nr;
 if(!cap) nr=0;
 else nr=1+numara_r(cap->next);
 return nr;}
```

ii. Să se scrie funcția nerecursivă pentru numărarea nodurilor unei liste simplu înlănțuite.

Funcția are ca parametru capul listei şi întoarce, prin numele ei, numărul de noduri ale listei.

```
int numara_i(TNOD* cap)
{int nr;
 nr=0;
 while(cap)
  {nr++;
   cap=cap->next;}
 return nr;}
```

iii. Să se scrie funcția pentru eliminarea tuturor nodurilor care conțin o anumită valoare dintr-o listă simplu înlănțuită.

Funcția are ca parametri adresa capului listei şi valoarea de identificare a nodurilor care trebuie şterse. Prin numele funcției se întoarce numărul de noduri şterse. Pentru ştergere este apelată funcția *sterg2* de la exercițiul 4.1.viii.

```
int elimina(TNOD **cap, float a)
{TNOD *p,*q;
 int n;
 n=0;
 while(!sterg2(cap,a))
  n++;
 return n;}
```

iv. Să se scrie funcția pentru sortarea unei liste simplu înlănțuite prin modificarea legăturilor între noduri.

Funcția are ca parametru capul listei şi întoarce, prin numele ei, noul cap al listei. Pentru a schimba două noduri consecutive între ele se apelează funcția *schimba*

care are ca parametri capul listei şi adresa primului din cele două noduri şi întoarce, prin numele ei, noul cap al listei.

```
TNOD* schimba(TNOD *cap, TNOD *p)
{TNOD *q,*r;
 if(p==cap){q=p->next;
            r=q->next;
            cap=q;q->next=p;
            p->next=r;}
 else{r=cap;
      while(r->next!=p)
        r=r->next;
      q=p->next;
      p->next=q->next;
      q->next=p;
      r->next=q;}
 q=cap;
 return cap;}

TNOD* sort(TNOD* cap)
{int er;
 TNOD *p,*q;
 er=1;
 if(cap)
 while(er)
  {er=0;
   for(p=cap;p->next;p=p->next)
      if(p->x>p->next->x){cap=schimba(cap,p);
                          er=1;}
  }
 return cap;}
```

v. Să se scrie funcţia pentru calculul sumei elementelor unui vector memorat ca listă simplu înlănţuită.

Funcţia are ca parametru capul listei şi întoarce, prin numele ei, suma elementelor.

```
float suma(TNOD *cap)
{float s=0;
 TNOD *p;
 p=cap;
 while(p)
  {s+=p->x;
   p=p->next;}
 return s;}
```

vi. Să se scrie funcţia pentru sortarea elementelor unui vector memorat ca listă simplu înlănţuită.

Funcţia are ca parametru capul listei.

```
void sort2(TNOD* cap)
{int er;
 float a;
 TNOD *p,*q;
 er=1;
 if(cap)
 while(er)
  {er=0;
   for(p=cap;p->next;p=p->next)
      if(p->x>p->next->x){a=p->x;
                          p->x=p->next->x;
                          p->next->x=a;
                          er=1;}
  }
}
```

vii. Să se scrie funcția pentru găsirea elementului minim şi a tuturor pozițiilor sale de apariție într-o listă simplu înlănțuită.

Funcția are ca parametri capul listei, adresa unde va scrie minimul şi vectorul în care va scrie pozițiile de apariție a minimului. Prin numele funcției se întoarce numărul de apariții ale minimului.

```
int minim(TNOD* cap, float *m,TNOD* poz[])
{TNOD* p;
 int n;
 n=0;
 if(cap)
  {*m=cap->x;
   while(p)
     {if(p->x<*m)
       {n=0;
        poz[n]=p;
        *m=p->x;}
      else if (p->x==*m)poz[n++]=p;
      p=p->next;}
  }
 return n;}
```

viii. Să se scrie funcția pentru interclasarea elementelor a doi vectori memorați în liste simplu înlănțuite. Se consideră că vectorii sînt sortați crescător.

Funcția are ca parametri capul primei liste şi capul celei de a doua liste. Prin numele funcției se întoarce capul listei rezultat.

```
TNOD* interc(TNOD* cap1,TNOD* cap2)
{TNOD *cap3, *p, *q, *r, *s;
 cap3=NULL;
 p=cap3;
 r=cap1; s=cap2;
 while(r&&s)
  {q=NEW;q->next=NULL;
   if(!cap3){cap3=q;p=q;}
```

```
   else{p->next=q;p=q;}
   if(r->x<s->x) {q->x=r->x;
                  r=r->next;}
   else {q->x=s->x;
         s=s->next;}
  }
 while(r)
  {q=NEW;q->next=NULL;
   if(!cap3){cap3=q;p=q;}
   else{p->next=q;p=q;}
   q->x=r->x;r=r->next;}
 while(s)
  {q=NEW;q->next=NULL;
   if(!cap3){cap3=q;p=q;}
   else{p->next=q;p=q;}
   q->x=s->x;s=s->next;}
  return cap3;}
```

ix. Să se scrie funcţia pentru crearea unei liste sortate din elementele unui vector.

Funcţia are ca parametri vectorul şi numărul său de elemente. Prin numele funcţiei se întoarce capul noii liste.

```
TNOD* creare(float x[], int n)
{int i;
 TNOD *cap,*p,*q;
 cap=NULL;
 for(i=0;i<n;i++)
   {q=NEW;
    q->x=x[i];
    if((!cap)||(cap->x>x[i]))
       {q->next=cap;
        cap=q;}
    else if(!cap->next){q->next=NULL;
                        cap->next=q;}
         else{p=cap;
         while((x[i]>p->next->x)&&(p->next))
           p=p->next;
         q->next=p->next;
         p->next=q;}
   }
 return cap;}
```

5. Lucrul cu fişiere

5.1 Fişiere text

În problemele din acest capitol se consideră, dacă nu se precizează altfel în enunţ, că fişierele sînt de dimensiune mare şi nu încap în memorie.

i. Să se scrie un program care copiază intrarea standard la ieşirea standard, pînă la apăsarea combinaţiei CTRL-Z.

```
#include<stdio.h>
#include<conio.h>
void main()
{ int c;
  while((c=getch())!=EOF)
  putch(c);}
```

ii. Să se scrie un program care listează la imprimanta conectată la portul paralel conţinutul fişierului text al cărui nume este preluat ca parametru în linia de comandă; dacă în linia de comandă nu se transmite nici un parametru, se consideră fişierul standard de intrare.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main(int argc,char *argv[])
{ FILE *f;
  int c,er;
  er=1;
  if(argc==1)
   {er=0;f=stdin;}
  else if(argc==2)
         if((f=fopen(argv[1],"r"))==0)
           printf("\nFisierul specificat nu poate fi deschis. (%s)",
                    argv[1]);
```

```
         else er=0;
     else printf("\nNumar gresit de parametri in linia de comanda.");
  if(!er)
   {while((c=getc(f))!=EOF)
    putc(c,stdprn);
  fclose(f);}
}
```

iii. Să se scrie programul care creează un fişier text în care memorează două matrice, astfel: pe prima linie numărul de linii şi numărul de coloane ale primei matrice, separate printr-un spaţiu; pe fiecare din liniile următoare, în ordine, elementele unei linii din matrice, separate prin cîte un spaţiu; în continuare a doua matrice, în aceeaşi formă.

```
#include<stdio.h>
void main()
{FILE *f;
 int m,n,i,j;
 float x;
 char s[20];
 printf("Nume fisier: ");gets(s);
 f=fopen(s,"w");
 printf("nr. linii a: ");scanf("%d",&m);
 printf("nr. col   a: ");scanf("%d",&n);
 fprintf(f,"%d %d\n",m,n);
 for(i=0;i<m;i++)
  {for(j=0;j<n;j++)
     {printf("a(%d,%d)=",i,j);
      scanf("%f",&x);
      fprintf(f,"%5.2f ",x);}
   fprintf(f,"\n");}
 printf("nr. linii b: ");scanf("%d",&m);
 printf("nr. col   b: ");scanf("%d",&n);
 fprintf(f,"%d %d\n",m,n);
 for(i=0;i<m;i++)
  {for(j=0;j<n;j++)
     {printf("b(%d,%d)=",i,j);
      scanf("%f",&x);
      fprintf(f,"%5.2f ",x);}
   fprintf(f,"\n");}
 fclose(f);}
```

iv. Să se scrie programul care înmulţeşte matricele aflate în fişierul creat la problema anterioară, dacă e posibil. Rezultatul se va memora în acelaşi fişier, în continuare, în aceeaşi formă. Dacă înmulţirea nu e posibilă, se va scrie ca rezultat un mesaj de eroare. Matricele sînt suficient de mici pentru a putea fi încărcate în memorie.

```
#include<stdio.h>
void main()
{FILE *f;
 int m,n,p,q,i,j,k;
```

```
 float x,a[20][20],b[20][20];
 char s[20];
 printf("Nume fisier: ");gets(s);
 if(!(f=fopen(s,"r"))) printf("\nFisierul nu poate fi deschis.");
 else{fscanf(f,"%d %d",&m,&n);
      for(i=0;i<m;i++)
        for(j=0;j<n;j++)
          fscanf(f,"%f",&a[i][j]);
      fscanf(f,"%d %d",&p,&q);
      for(i=0;i<p;i++)
        for(j=0;j<q;j++)
      fscanf(f,"%f",&b[i][j]);
      if(n!=p) printf("\nNu se pot inmulti.");
      else{f=fopen(s,"a");
           fprintf(f,"%d %d\n", m,p);
           for(i=0;i<m;i++)
            {for(j=0;j<q;j++)
              {x=0;
               for(k=0;k<n;k++)
               x+=a[i][k]*b[k][j];
               fprintf(f,"%5.2f ",x);}
             fprintf(f,"\n");}
          }
      fclose(f);}
}
```

v. Să se scrie programul care afişează pe ecran rezultatul de la problema anterioară.

```
#include<stdio.h>
#define MANY 1000
void main()
{FILE *f;
 int m,n,i,j;
 float x;
 char s[20];
 printf("Nume fisier: ");gets(s);
 if(!(f=fopen(s,"r"))) printf("\nFisierul nu poate fi deschis.");
 else{fscanf(f,"%d %d",&m,&n);
      fgets(s,MANY,f);
      for(i=0;i<m;i++)
        fgets(s,MANY,f);
      fscanf(f,"%d %d",&m,&n);
      fgets(s,MANY,f);
      for(i=0;i<m;i++)
        fgets(s,MANY,f);
      fscanf(f,"%d %d",&m,&n);
      for(i=0;i<m;i++)
       {for(j=0;j<n;j++)
         {fscanf(f,"%f",&x);
          printf("\t%5.2f",x);}
          printf("\n");}
      fclose(f);}
}
```

vi. Să se scrie programul care memorează într-un fişier text date despre studenții unei facultăți. Pe fiecare linie se memorează: numărul matricol, numele, prenumele, anul, grupa, cele 10 note obținute la examene. Datele sînt separate prin cîte un spațiu. Acolo unde nu se cunoaşte încă nota se va introduce valoarea 0.

```
#include<stdio.h>
void main()
{FILE *f;
 char nume[30],s[20];
 int nr,an,gr,n[10],i;
 printf("Nume fisier: ");
 gets(s);
 f=fopen(s,"w");
 printf("Nr. matricol: ");
 scanf("%d",&nr);
 while(!feof(stdin))
 {fflush(stdin);
  printf("Nume si prenume: ");gets(nume);
  printf("An: ");scanf("%d",&an);
  printf("Grupa: ");scanf("%d",&gr);
  fprintf(f,"%4d %30s %2d %2d ",nr,nume,an,gr);
  for(i=0;i<10;i++)
    {printf("nota %d: ",i+1);
     scanf("%d",&n[i]);
     fprintf(f,"%2d ",n[i]);}
  fprintf(f,"\n");
  printf("Nr. matricol: ");
  scanf("%d",nr);}
  fclose(f);}
```

vii. Să se scrie programul care calculează media studenților integralişti şi cea mai mare dintre mediile studenților integralişti din fişierul creat anterior.

```
#include<stdio.h>
void main()
{FILE *f;
 char nume[30],s[200];
 int nr,an,gr,n[10],i,e,p;
 float m,max,m1;
 printf("Nume fisier: ");
 gets(s);
 if(!(f=fopen(s,"r")))printf("Nu se poate deschide fisierul.");
 else {p=0;max=0;m1=0;
       fscanf(f,"%d",&nr);
       while(!feof(f))
        {fscanf(f,"%s %d %d",nume,&an,&gr);
         e=1;m=0;
        for(i=0;i<10;i++)
          {fscanf(f,"%d",&n[i]);
           if(n[i]<5)e=0;
           m+=n[i];}
```

```
        if(e){m/=10;
              p++;
              m1+=m;
              if(m>max)max=m;}
        fscanf(f,"%d",&nr);}
       printf("\nmedia: %5.2f",m1/p);
       printf("\nmedia maxima: %5.2f", max);}
 fclose(f);}
```

5.2 Fişiere binare

5.2.1 Vectori memoraţi în fişiere binare

Pentru lucrul cu vectori memoraţi în fişiere binare s-a adoptat următorul mod de organizare: se memorează cîte un element al vectorului în fiecare articol, în ordine lexicografică. Se vor folosi vectori cu elemente reale.

i. Să se scrie programul care memorează elementele unui vector într-un fişier binar. Datele se preiau de la tastatură, sfîrşitul introducerii fiind marcat standard.

```
#include<stdio.h>
void main(void)
{FILE *f;
 float x;
 char nume[20];
 printf("\nfisier=");
 gets(nume);
 f=fopen(nume,"wb");
 printf("x=");
 scanf("%f",&x);
 while(!feof(stdin))
  {fwrite(&x,sizeof(float),1,f);
   printf("x=");
   scanf("%f",&x);}
 fclose(f);}
```

ii. Să se scrie programul care afişează pe ecran şi scrie într-un fişier text vectorul memorat în fişier la problema anterioară.

```
#include<stdio.h>
#include<conio.h>
void main(void)
{FILE *f,*g;
 float x;
 char nume[20];
 printf("\nfisier=");
```

```
 gets(nume);
 if(f=fopen(nume,"rb"))
  {g=fopen("imagine.txt","w");
   printf("\n");
   fread(&x,sizeof(float),1,f);
   while(!feof(f))
    {printf("%7.2f  ",x);
     fprintf(g,"%7.2f",x);
     fread(&x,sizeof(float),1,f);}
   fclose(f);
   fclose(g);}
 getch();}
```

iii. Să se scrie programul care afişează amplitudinea şi media aritmetică a elementelor unui vector memorat într-un fişier binar.

```
#include<stdio.h>
#include<conio.h>
void main(void)
{FILE *f;
 float x,m,max,min;
 int n;
 char nume[20];
 printf("\nfisier=");
 gets(nume);
 if(f=fopen(nume,"rb"))
 {m=0;
  n=0;
  if(fread(&x,sizeof(float),1,f)!=sizeof(float))
   {max=min=x;
    fread(&x,sizeof(float),1,f);
    while(!feof(f))
     {if(x<min)min=x;
      if(x>max)max=x;
      m+=x;n++;
      fread(&x,sizeof(float),1,f);}
    printf("Amplitudine: %7.2f, media: %7.2f",max-min,m/n);}
  else printf("\nFisier gol");
 fclose(f);}
 getch();}
```

iv. Să se scrie programul care calculează produsul vectorial dintre doi vectori memorați în fişiere binare. Rezultatul va fi memorat în alt fişier.

```
#include<stdio.h>
int nrart(FILE *f, int l)
{long p;
 int n;
 p=ftell(f);
 fseek(f,0,2);
 n=ftell(f)/l;
 fseek(f,0,p);
 return n;}
```

```
void main()
{FILE *f,*g,*h;
 float x,y,z;
 char s1[20],s2[20];
 printf("\nPrimul fisier: ");gets(s1);
 printf("Al doilea fis: ");gets(s2);
 if(!(f=fopen(s1,"rb")))printf("Fisierul %s nu poate fi deschis.",
                                 s1);
 else if(!(g=fopen(s2,"rb")))printf("Fisierul %s nu poate fi deschis",
                                    s2);
      else {if(nrart(f,sizeof(float))!=nrart(g,sizeof(float)))
              printf("Numar diferit de articole.");
            else {printf("Fisier rezultat: ");
                  gets(s1);
                  h=fopen(s1,"wb");
                  fread(&x,sizeof(float),1,f);
                  fread(&y,sizeof(float),1,g);
                  while(!feof(f))
                   {z=x*y;
                    fwrite(&z,sizeof(float),1,h);
                    fread(&x,sizeof(float),1,f);
                    fread(&y,sizeof(float),1,g);}
                  fclose(h);}
            fclose(f);
            fclose(g);}
}
```

v. Să se scrie programul care afişează produsul scalar dintre doi vectori memorați în fişiere binare.

```
#include<stdio.h>
int nrart(FILE *f, int l)
{long p;
 int n;
 p=ftell(f);
 fseek(f,0,2);
 n=ftell(f)/l;
 fseek(f,0,p);
 return n;}

void main()
{FILE *f,*g;
 float x,y,z;
 char s1[20],s2[20];
 printf("\nPrimul fisier: ");gets(s1);
 printf("Al doilea fis: ");gets(s2);
 if(!(f=fopen(s1,"rb")))printf("Fisierul %s nu poate fi deschis.",s1);
 else if(!(g=fopen(s2,"rb")))printf("Fisierul %s nu poate fi deschis.",
                                    s2);
      else {if(nrart(f,sizeof(float))!=nrart(g,sizeof(float)))
              printf("Numar diferit de articole.");
```

```
            else {z=0;
            fread(&x,sizeof(float),1,f);
            fread(&y,sizeof(float),1,g);
            while(!feof(f))
             {z+=x*y;
              fread(&x,sizeof(float),1,f);
              fread(&y,sizeof(float),1,g);}
            printf("Rezultat: %7.2f",z);}
            fclose(f);
            fclose(g);}
}
```

vi. Să se scrie programul care sortează crescător un vector memorat într-un fişier binar, prin metoda bulelor.

```
#include<stdio.h>
#include<conio.h>
void main(void)
{FILE *f;
 float x,y;
 int i,m,n;
 char nume[20];
 printf("\nFisier="); gets(nume);
 if(f=fopen(nume,"rb+"))
  {fseek(f,0,SEEK_END);
   n=ftell(f)/sizeof(float);
   m=1;
   while(m)
    {rewind(f);
     m=0;
     for(i=0;i<n;i++)
      {fseek(f,i*sizeof(float),SEEK_SET);
       fread(&x,sizeof(float),1,f);
       fread(&y,sizeof(float),1,f);
       if(x>y)
        {fseek(f,ftell(f)-2*sizeof(float),SEEK_SET);
         fwrite(&y,sizeof(float),1,f);
         fwrite(&x,sizeof(float),1,f);
         m=1;}
      }
    }
   fclose(f);}
 else printf("Fisierul %s nu poate fi deschis.", nume);}
```

vii. Să se scrie programul care sortează crescător un vector memorat într-un fişier binar, prin metoda selecției.

```
#include<stdio.h>
int nrart(FILE *f, int l)
{long p;
 int n;
 p=ftell(f);
 fseek(f,0,2);
```

```
 n=ftell(f)/l;
 fseek(f,0,p);
 return n;}
void main()
{FILE *f;
 char s[20];
 int n,i,p,j;
 float x,y,z;
 printf("\nFisier: ");
 gets(s);
 if(!(f=fopen(s,"rb+")))printf("\nFisierul %s nu poate fi deschis.",s);
 else{n=nrart(f,sizeof(float));
      for(i=0;i<n-1;i++)
       {p=i;
        fseek(f,p*sizeof(float),SEEK_SET);
        fread(&x,sizeof(float),1,f);
        z=x;
        for(j=i+1;j<n;j++)
         {fseek(f,j*sizeof(float),SEEK_SET);
          fread(&y,sizeof(float),1,f);
          if(x>y){p=j;x=y;}
        }
      fseek(f,p*sizeof(float),SEEK_SET);
      fwrite(&z,sizeof(float),1,f);
      fseek(f,i*sizeof(float),SEEK_SET);
      fwrite(&x,sizeof(float),1,f);}
      fclose(f);}
}
```

viii. Să se scrie programul care sortează crescător un vector memorat într-un fişier binar, prin metoda QuickSort.

```
#include<stdio.h>
int nrart(FILE *f, int l)
{long p;
 int n;
 p=ftell(f);
 fseek(f,0,2);
 n=ftell(f)/l;
 fseek(f,0,p);
 return n;}



void quicksort(FILE *f, int inf,int sup)
{int i,j,ii,jj;
 float x,y;
 if(inf<sup)
  {i=inf;j=sup;ii=0;jj=-1;
   while(i<j)
    {fseek(f,i*sizeof(float),SEEK_SET);
     fread(&x,sizeof(float),1,f);
     fseek(f,j*sizeof(float),SEEK_SET);
     fread(&y,sizeof(float),1,f);
```

```
     if(x>y)
      {fseek(f,i*sizeof(float),SEEK_SET);
       fwrite(&y,sizeof(float),1,f);
       fseek(f,j*sizeof(float),SEEK_SET);
       fwrite(&x,sizeof(float),1,f);
       if(ii){ii=0;jj=-1;}
       else{ii=1;jj=0;}
      }
     i+=ii;
     j+=jj;
    }
   quicksort(f,inf,i-1);
   quicksort(f,i+1,sup);
  }
}
void main()
{FILE *f;
 int n;
 char s[20];
 printf("\nFisier: ");
 gets(s);
 if(!(f=fopen(s,"rb+")))printf("Fisierul %s nu poate fi deschis.",s);
 else{n=nrart(f,sizeof(float));
      quicksort(f,0,n-1);
      fclose(f);}
}
```

ix. Să se scrie programul care compactează un vector memorat într-un fişier binar, prin eliminarea dublurilor.

```
#include<stdio.h>
void main()
{FILE *f,*g;
 char s[20],e;
 float x,y;
 printf("\nFisier: ");
 gets(s);
 if(!(f=fopen(s,"rb"))) printf("Fisierul %s nu poate fi deschis.",s);
 else{g=fopen("temp.dat","wb+");
      fread(&x,sizeof(float),1,f);
      while(!feof(f))
       {rewind(g); e=0;
        fread(&y,sizeof(float),1,g);
        while((!feof(g))&&(!e))
         {if(x==y)e=1;
          fread(&y,sizeof(float),1,g);}
        if(!e){fseek(g,0,SEEK_END);
          fwrite(&x,sizeof(float),1,g);}
       fread(&x,sizeof(float),1,f);}
      fclose(f);
      fclose(g);
      unlink(s);
      rename("temp.dat",s);}
}
```

x. Să se scrie programul care numără frecvența de apariție a fiecărei valori dintr-un vector memorat într-un fişier binar. Rezultatul va fi memorat într-un fişier text astfel: pe fiecare linie, separate printr-un spațiu, se vor scrie valoarea, numărul de apariții şi frecvența relativă de apariție. Fişierul inițial nu se va sorta.

```
#include<stdio.h>
typedef struct{float x;
               int n;}ART;
void main()
{FILE *f,*g,*h;
 float x;
 ART a;
 int n,e;
 char s1[20],s2[20];
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("Fisierul %s nu poate fi deschis.",s1);
 else{printf("Fisierul rezultat:");
      gets(s2);
      g=fopen("temp.dat","wb+");
      n=0;
      fread(&x,sizeof(float),1,f);
      while(!feof(f))
       {rewind(g);
        e=0;n++;
        fread(&a,sizeof(ART),1,g);
        while((!feof(g))&&(!e))
         {if(a.x==x){a.n++;
                     e=1;
                     fseek(g,ftell(g)-sizeof(ART),SEEK_SET);
                     fwrite(&a,sizeof(ART),1,g);}
                     fread(&a,sizeof(ART),1,g);}
        if(!e){a.x=x;a.n=1;fwrite(&a,sizeof(ART),1,g);}
        fread(&x,sizeof(float),1,f);}
      fclose(f);
      h=fopen(s2,"w");
      rewind(g);
      fprintf(h,"\n\tNumar   \tAp \tFrecv");
      fread(&a,sizeof(ART),1,g);
      while(!feof(g))
       {fprintf(h,"\n\t%7.2f \t%d \t%7.2f",a.x,a.n,a.n/(float)n);
        fread(&a,sizeof(ART),1,g);}
      fclose(g);fclose(h);
      unlink("temp.dat");}
}
```

xi. Să se scrie funcția care determină dacă un vector memorat într-un fişier binar este sortat strict crescător, strict descrescător, constant sau amestecat.

```
#include<stdio.h>b
int nrart(FILE *f, int l)
```

```
{long p;
 int n;
 p=ftell(f);
 fseek(f,0,2);
 n=ftell(f)/l;
 fseek(f,0,p);
 return n;}
void main()
{FILE *f;
 int sc,sd,c;
 char s[20];
 float x,y;
 sc=sd=c=1;
 printf("\nFisier: ");
 gets(s);
 if(!(f=fopen(s,"rb")))printf("\nFisierul %s nu poate fi deschis.",s);
 else{fread(&x,sizeof(float),1,f);
      if(feof(f))printf("\nFisier gol");
      else{fread(&y,sizeof(float),1,f);
           if(feof(f))printf("\nFisier cu un singur element");
           else{while((!feof(f))&&(sc||sd||c))
                 {if(x>=y)sc=0;
                  if(x<=y)sd=0;
                  if(x!=y)c=0;
                  x=y;
                  fread(&y,sizeof(float),1,f);}
                if(sc)printf("\nSortat crescator.");
                else if(sd)printf("\nSortat descrescator.");
                     else if(c)printf("\nConstant");
                          else printf("\nAmestecat");}
          }
      fclose(f);}
}
```

xii. Să se scrie programul care interclasează doi vectori sortați crescător, aflați în fişiere binare. Rezultatul se va memora în alt fişier.

```
#include<stdio.h>
void main()
{FILE *f,*g,*h;
 float x,y;
 char s1[20],s2[20],s3[30];
 printf("\nFisier 1:");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis.",s1);
 else{printf("\nFisier 2:");gets(s2);
      if(!(g=fopen(s2,"rb")))printf("\nFisierul %s nu poate fi deschis.",
                                    s2);
      else {printf("\nFisier rezultat: ");gets(s3);
            h=fopen(s3,"wb");
            fread(&x,sizeof(float),1,f);
            fread(&y,sizeof(float),1,g);
            while((!feof(f))&&(!feof(g)))
              if(x<y){fwrite(&x,sizeof(float),1,h);
                      fread(&x,sizeof(float),1,f);}
```

```
              else{fwrite(&y,sizeof(float),1,h);
                   fread(&y,sizeof(float),1,g);}
            while(!feof(g))
             {fwrite(&y,sizeof(float),1,h);
              fread(&y,sizeof(float),1,g);}
            while(!feof(f))
             {fwrite(&x,sizeof(float),1,h);
              fread(&x,sizeof(float),1,f);}
            fclose(g);}
      fclose(f);}
}
```

5.2.2 Matrice memorate în fişiere binare

Pentru lucrul cu matrice memorate în fişiere binare s-a adoptat următorul mod de organizare: primul articol conţine numărul de linii ale matricei, iar următoarele articole conţin, în ordine, cîte un element al matricei, în ordine lexicografică. Se va lucra cu matrice cu elemente reale. Numărul de linii este întreg (primul articol are altă dimensiune).

i. Să se scrie programul care memorează într-un fişier binar o matrice introdusă de la tastatură.

```
#include<stdio.h>
void main()
{FILE* f; float x;
 int m,n,i,j;
 char s[20];
 printf("\nNume fisier="); gets(s);
 f=fopen(s,"wb");
 printf("m=");scanf("%d",&m);
 printf("n=");scanf("%d",&n);
 fwrite(&m,sizeof(int),1,f);
 for(i=0;i<m;i++)
   for(j=0;j<n;j++)
    {printf("a(%d,%d)=",i,j);
     scanf("%f",&x);
     fwrite(&x,sizeof(float),1,f);}
 fclose(f);}
```

ii. Să se scrie programul care afişează pe ecran şi într-un fişier text matricea memorată într-un fişier binar. Se consideră că matricea are dimensiuni suficient de mici pentru a încăpea pe ecran.

```
#include<stdio.h>
void main()
{FILE *f,*g;
 float x;
 int m,n,i,j;
 char s[20];
```

```
 printf("\nNume fisier=");
 gets(s);
 if(!(f=fopen(s,"rb")))printf("Fisierul %s nu poate fi deschis.",s);
 else{g=fopen("Imag_mat.txt","w");
      fseek(f,0,SEEK_END);
      n=ftell(f)-sizeof(int);
      rewind(f);
      fread(&m,sizeof(int),1,f);
      n=n/(m*sizeof(float));
      for(i=0;i<m;i++)
       {printf("\n");
        fprintf(g,"\n");
        for(j=0;j<n;j++)
         {fread(&x,sizeof(float),1,f);
          printf("%7.2f ",x);
          fprintf(g,"%7.2f ",x);}
       }
      fclose(f);
      fclose(g);}
}
```

iii. Să se scrie programul care calculează produsul dintre două matrice memorate în fişiere binare. Rezultatul se va memora în alt fişier.

```
#include<stdio.h>
void main()
{FILE *f,*g,*h;
 float x,y,z;
 int m,n,p,q,i,j,k;
 char s1[20],s2[20],s3[20];
 printf("\nNume fisier1=");
 gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis.",s1);
 else{printf("Nume fisier1=");
      gets(s2);
      if(!(g=fopen(s2,"rb")))printf("\nFisierul %s nu poate fi deschis",
                                    s2);
      else{printf("Rezultat: ");gets(s3);
           h=fopen(s3,"wb");
           fseek(f,0,SEEK_END);
           n=ftell(f)-sizeof(int);
           rewind(f);
           fread(&m,sizeof(int),1,f);
           n=n/(m*sizeof(float));
           fseek(g,0,SEEK_END);
           q=ftell(g)-sizeof(int);
           rewind(g);
           fread(&p,sizeof(int),1,g);
           q=q/(p*sizeof(float));
           if(n!=p)printf("\nMatricele nu se pot inmulti.");
           else{fwrite(&m,sizeof(int),1,h);
                for(i=0;i<m;i++)
                  for(j=0;j<q;j++)
                   {z=0;
```

```
                    for(k=0;k<n;k++)
                     {fseek(f,(i*n+k)*sizeof(float)+sizeof(int),0);
                      fread(&x,sizeof(float),1,f);
                      fseek(g,(k*q+j)*sizeof(float)+sizeof(int),0);
                      fread(&y,sizeof(float),1,g);
                      z+=x*y;}
                    fwrite(&z,sizeof(float),1,h);}
                }
           fclose(h);
           fclose(g);}
      fclose(f);}
}
```

iv. Să se scrie programul care determină liniile dintr-o matrice memorată într-un fişier binar, care au elementele în ordine strict crescătoare.

```
#include<stdio.h>
#define POZ(i,j,n) sizeof(int)+((i)*(n)+j)*sizeof(float)
void main()
{FILE *f,*g;
 char s1[20],s2[20];
 float x,y;
 int m,n,i,j,e;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis.",s1);
 else{printf("Rezultat (text):");
      gets(s2);
      g=fopen(s2,"w");
      fprintf(g,"Liniile cautate sint: ");
      fseek(f,0,SEEK_END);
      n=ftell(f)-sizeof(int);
      rewind(f);
      fread(&m,sizeof(int),1,f);
      n=n/(m*sizeof(float));
      for(i=0;i<m;i++)
       {fseek(f,POZ(i,0,n),SEEK_SET);
        fread(&x,sizeof(float),1,f);
        e=1;
        for(j=1;(j<n)&&e;j++)
         {fseek(f,POZ(i,j,n),SEEK_SET);
          fread(&y,sizeof(float),1,f);
          if(x>=y)e=0;
            x=y;}
        if(e)fprintf(g,"\t%d",i);}
      fclose(f);
      fclose(g);}
}
```

v. Să se scrie programul care determină coloanele dintr-o matrice memorată într-un fişier binar, care au elementele în ordine strict crescătoare.

```
#include<stdio.h>
#define POZ(i,j,n) sizeof(int)+((i)*(n)+j)*sizeof(float)
```

```
void main()
{FILE *f,*g;
 char s1[20],s2[20];
 float x,y;
 int m,n,i,j,e;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis.",s1);
 else{printf("Rezultat (text):");
      gets(s2);
      g=fopen(s2,"w");
      fprintf(g,"Coloanele cautate sint: ");
      fseek(f,0,SEEK_END);
      n=ftell(f)-sizeof(int);
      rewind(f);
      fread(&m,sizeof(int),1,f);
      n=n/(m*sizeof(float));
      for(i=0;i<n;i++)
       {fseek(f,POZ(0,i,n),SEEK_SET);
        fread(&x,sizeof(float),1,f);
        e=1;
        for(j=1;(j<m)&&e;j++)
         {fseek(f,POZ(j,i,n),SEEK_SET);
          fread(&y,sizeof(float),1,f);
          if(x>=y)e=0;
          x=y;}
        if(e)fprintf(g,"\t%d",i);
       }
      fclose(f);
      fclose(g);}
}
```

vi. Să se scrie programul care determină amplitudinea elementelor de pe diagonala principală, respectiv secundară a unei matrice memorate într-un fişier binar. Nu se ştie dacă matricea este pătrată.

```
#include<stdio.h>
#define POZ(i,j,n) sizeof(int)+((i)*(n)+j)*sizeof(float),SEEK_SET
void main()
{FILE *f;
 int m,n,i;
 float maxp,maxs,minp,mins,x;
 char s[20];
 printf("\nFisier: ");gets(s);
 if(!(f=fopen(s,"rb")))printf("\nFisierul %s nu poate fi deschis",s);
 else{fseek(f,0,SEEK_END);
      n=ftell(f)-sizeof(int);
      rewind(f);
      fread(&m,sizeof(int),1,f);
      n=n/(m*sizeof(float));
      if(m!=n) printf("\nFisierul nu contine o matrice patrata.");
      else{fseek(f,POZ(0,0,n));
           fread(&maxp,sizeof(float),1,f);
           minp=maxp;
           fseek(f,POZ(0,0,n-1));
```

```
            fread(&maxs,sizeof(float),1,f);
            for(i=1;i<n;i++)
             {fseek(f,POZ(i,i,n));
              fread(&x,sizeof(float),1,f);
              if(x>maxp)maxp=x;
              if(x<minp)minp=x;
              fseek(f,POZ(i,n-i-1,n));
              fread(&x,sizeof(float),1,f);
              if(x>maxs)maxs=x;
              if(x<mins)mins=x;}
            printf("\nAmplitudine d.p.: %7.2f",maxp-minp);
            printf("\nAMplitudine d.s.: %7.2f",maxs-mins);;}
      fclose(f);}
}
```

vii. Să se scrie programul care numără zerourile de sub diagonala secundară a unei matrice pătrate memorate într-un fişier binar.

```
#include<stdio.h>
#define POZ(i,j,n) sizeof(int)+((i)*(n)+j)*sizeof(float),SEEK_SET
void main()
{FILE *f;
 int m,n,i,j;
 float x;
 char s[20];
 printf("\nFisier: ");gets(s);
 if(!(f=fopen(s,"rb")))printf("\nFisierul %s nu poate fi deschis",s);
 else{fseek(f,0,SEEK_END);
      n=ftell(f)-sizeof(int);
      rewind(f);
      fread(&m,sizeof(int),1,f);
      n=n/(m*sizeof(float));
      if(m!=n) printf("\nFisierul nu contine o matrice patrata.");
      else{n=0;
           for(i=0;i<m;i++)
             for(j=0;j<i-1;j++)
              {fseek(f,POZ(i,j,m));
               fread(&x,sizeof(float),1,f);
               if(!x)n++;}
           printf("\n Sub d.p. sint %d zerouri.",n);}
      fclose(f);}
}
```

viii. Să se scrie programul care determină elementul minim din triunghiul aflat deasupra ambelor diagonale ale unei matrice pătrate memorate într-un fişier binar.

```
#include<stdio.h>
#define POZ(i,j,n) sizeof(int)+((i)*(n)+j)*sizeof(float),SEEK_SET
void main()
{FILE *f;
 int m,n,i,j;
 float min,x;
```

```
 char s[20];
 printf("\nFisier: ");gets(s);
 if(!(f=fopen(s,"rb")))printf("\nFisierul %s nu poate fi deschis",s);
 else{fseek(f,0,SEEK_END);
      n=ftell(f)-sizeof(int);
      rewind(f);
      fread(&m,sizeof(int),1,f);
      n=n/(m*sizeof(float));
      if(m!=n) printf("\nFisierul nu contine o matrice patrata.");
      else{fseek(f,POZ(0,1,m));
           fread(&min,sizeof(float),1,f);
           for(i=0;i<m;i++)
             for(j=i+1;j<n-i-1;j++)
              {fseek(f,POZ(i,j,m));
               fread(&x,sizeof(float),1,f);
               if(x<min)min=x;}
           printf("\nMinim: %7.2f",min);}
      fclose(f);}
}
```

ix. Să se scrie programul care determină elementul minim din triunghiul aflat sub ambele diagonale ale unei matrice pătrate memorate într-un fişier binar.

```
#include<stdio.h>
#define POZ(i,j,n) sizeof(int)+((i)*(n)+j)*sizeof(float),SEEK_SET
void main()
{FILE *f;
 int m,n,i,j;
 float min,x;
 char s[20];
 printf("\nFisier: ");gets(s);
 if(!(f=fopen(s,"rb")))printf("\nFisierul %s nu poate fi deschis",s);
 else{fseek(f,0,SEEK_END);
      n=ftell(f)-sizeof(int);
      rewind(f);
      fread(&m,sizeof(int),1,f);
      n=n/(m*sizeof(float));
      if(m!=n) printf("\nFisierul nu contine o matrice patrata.");
      else{fseek(f,POZ(0,1,m));
           fread(&min,sizeof(float),1,f);
           for(i=0;i<m;i++)
             for(j=n-i;j<i;j++)
              {fseek(f,POZ(i,j,m));
               fread(&x,sizeof(float),1,f);
               if(x<min)min=x;}
           printf("\nMinim: %7.2f",min);}
      fclose(f);}
}
```

x. Să se scrie programul care calculează produsul dintre o matrice şi un vector, ambele memorate în fişiere binare. Rezultatul se va memora în alt fişier.

```
#include<stdio.h>
#define POZ(i,j,n) sizeof(int)+((i)*(n)+(j))*sizeof(float),SEEK_SET
void main()
{FILE *f,*g,*h;
 int m,n,p,i,j;
 float x,y,z;
 char s1[20],s2[20],s3[20];
 printf("\nMatricea: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nVectorul: ");gets(s2);
      if(!(g=fopen(s2,"rb")))printf("\nFisierul %s nu poate fi deschis",
                                    s2);
      else{fseek(f,0,SEEK_END);
           n=ftell(f)-sizeof(int);
           rewind(f);
           fread(&m,sizeof(int),1,f);
           n=n/(m*sizeof(float));
           fseek(g,0,SEEK_END);
           p=ftell(g)/sizeof(float);
           if(n!=p) printf("\nInmultire imposibila");
           else{printf("\nRezultat: ");gets(s3);
                h=fopen(s3,"wb");
                for(i=0;i<m;i++)
                 {z=0;
                  for(j=0;j<n;j++)
                   {fseek(f,POZ(i,j,n));
                    fread(&x,sizeof(float),1,f);
                    fseek(g,j*sizeof(float),SEEK_SET);
                    fread(&y,sizeof(float),1,g);
                    z+=x*y;}
                 fwrite(&z,sizeof(float),1,h);}
                fclose(h);}
           fclose(g);}
      fclose(f);}

}
```

xi. Să se scrie programul care calculează produsul dintre un vector şi o matrice, ambele memorate în fişiere binare. Rezultatul se va memora în alt fişier.

```
#include<stdio.h>
#define POZ(i,j,n) sizeof(int)+((i)*(n)+(j))*sizeof(float),SEEK_SET
void main()
{FILE *f,*g,*h;
 int m,n,p,i,j;
 float x,y,z;
 char s1[20],s2[20],s3[20];
 printf("\nVectorul: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nMatricea: ");gets(s2);
      if(!(g=fopen(s2,"rb")))printf("\nFisierul %s nu poate fi deschis",
                                    s2);
      else{fseek(g,0,SEEK_END);
```

```
            n=ftell(g)-sizeof(int);
            rewind(g);
            fread(&m,sizeof(int),1,g);
            n=n/(m*sizeof(float));
            fseek(f,0,SEEK_END);
            p=ftell(f)/sizeof(float);
            if(m!=p) printf("\nInmultire imposibila");
            else{printf("\nRezultat: ");gets(s3);
                 h=fopen(s3,"wb");
                 for(i=0;i<n;i++)
                  {z=0;
                   for(j=0;j<m;j++)
                    {fseek(g,POZ(j,i,n));
                     fread(&x,sizeof(float),1,g);
                     fseek(f,j*sizeof(float),SEEK_SET);
                     fread(&y,sizeof(float),1,f);
                     z+=x*y;}
                   fwrite(&z,sizeof(float),1,h);}
                 fclose(h);}
            fclose(g);}
      fclose(f);}
}
```

xii. Să se scrie programul care construieşte o matrice prin eliminarea liniilor formate numai din zerouri ale unei matrice memorate într-un fişier binar.

```
#include<stdio.h>
#define POZ(i,j,n) sizeof(int)+((i)*(n)+(j))*sizeof(float),SEEK_SET

void main()
{FILE *f,*g;
 int m,n,p,e,i,j;
 float x;
 char s1[20],s2[20];
 printf("\nMatricea: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nmatricea noua: ");gets(s2);
      g=fopen(s2,"wb+");
      fseek(f,0,SEEK_END);
      n=ftell(f)-sizeof(int);
      rewind(f);
      fread(&m,sizeof(int),1,f);
      n=n/(m*sizeof(float));
      fwrite(&m,sizeof(int),1,g);
      p=m;
      for(i=0;i<m;i++)
       {e=1;
        fseek(f,POZ(i,0,n));
        for(j=0;(j<n)&&e;j++)
         {fread(&x,sizeof(float),1,f);
          if(x)e=0;}
        if(!e){fseek(f,POZ(i,0,n));
               for(j=0;j<n;j++)
                {fread(&x,sizeof(float),1,f);
```

```
                fwrite(&x,sizeof(float),1,g);}
              }
        else p--;}
      rewind(g);
      fwrite(&p,sizeof(int),1,g);
      fclose(g);
      fclose(f);}
}
```

xiii. Să se scrie programul care determină dacă o matrice aflată într-un fişier binar este sau nu rară. O matrice este rară dacă are maxim 30% din elemente diferite de 0.

```
#include<stdio.h>
void main()
{FILE *f;
 int m,n,e;
 float x;
 char s[20];
 printf("\nMatricea: ");gets(s);
 if(!(f=fopen(s,"rb")))printf("\nFisierul %s nu poate fi deschis",s);
 else{e=0;
      fseek(f,0,SEEK_END);
      n=(ftell(f)-sizeof(int))/sizeof(float);
      rewind(f);
      fread(&m,sizeof(int),1,f);
      fread(&x,sizeof(float),1,f);
      while(!feof(f))
       {if(!x)e++;
        fread(&x,sizeof(float),1,f);}
        if((float)e/n>=0.7)printf("\nMatrice rara");
        else printf("\nNu e matrice rara");
        fclose(f);}
}
```

xiv. Să se scrie programul care compactează un fişier binar conţinînd o matrice rară, folosind memorarea matricei în 3 vectori (linie, coloană, valoare). Cei 3 vectori se vor memora într-un fişier binar, intercalaţi (linie, coloană, valoare; linie, coloană, valoare;...).

```
#include<stdio.h>
typedef struct ART{int l;
                   int c;
                   float x;
                  }ELEMENT;
void main()
{FILE *f,*g;
 ELEMENT a;
 int m,n,e,i,j;
 float x;
 char s[20],s2[20];
 printf("\nMatricea: ");gets(s);
 if(!(f=fopen(s,"rb")))printf("\nFisierul %s nu poate fi deschis",s);
```

```
 else{fseek(f,0,SEEK_END);
      n=(ftell(f)-sizeof(int))/sizeof(float);
      rewind(f);
      fread(&m,sizeof(int),1,f);
      n=n/m;
      printf("\nRezultat: ");gets(s2);
      g=fopen(s2,"wb");
      for(i=0;i<m;i++)
        for(j=0;j<n;j++)
         {fread(&x,sizeof(float),1,f);
          if(x){a.x=x;
                a.l=i;
                a.c=j;
                fwrite(&a,sizeof(ELEMENT),1,g);}
         }
      fclose(f);
      fclose(g);}
}
```

xv. Să se scrie programul care sortează prima linie a unei matrice memorate într-un fişier binar, fără a schimba structura coloanelor.

```
#include<stdio.h>
#define POZ(i,j,n) sizeof(int)+((i)*(n)+(j))*sizeof(float),SEEK_SET
void main()
{FILE *f;
 char s[20];
 int m,n,i,j,k,e; float x,y;
 printf("\nMatricea: "); gets(s);
 if(!(f=fopen(s,"rb+")));
 else{fseek(f,0,SEEK_END);
      n=ftell(f)-sizeof(int);
      rewind(f); fread(&m,sizeof(int),1,f);
      n=n/(m*sizeof(float));
      e=1;
      while(e)
      {e=0;
       for(j=0;j<n-1;j++)
        {fseek(f,POZ(0,j,n));
         fread(&x,sizeof(float),1,f);
         fread(&y,sizeof(float),1,f);
         if(x>y)
          {for(k=0;k<m;k++)
            {fseek(f,POZ(k,j,n));
             fread(&x,sizeof(float),1,f);
             fread(&y,sizeof(float),1,f);
             fseek(f,POZ(k,j,n));
             fwrite(&y,sizeof(float),1,f);
             fwrite(&x,sizeof(float),1,f);}
           e=1;}
        }
      }
      fclose(f);}
}
```

5.2.3 Fişiere organizate secvenţial

i. Să se scrie programul care creează un fişier secvenţial cu date despre studenţii unei facultăţi. Articolele au următoarea structură: număr matricol, nume, anul, grupa, numărul de note, notele (maxim 15). Datele se preiau de la tastatură, sfîrşitul introducerii fiind marcat standard. Acolo unde nu se cunoaşte încă nota se va introduce valoarea 0.

```
#include<stdio.h>
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f;
 char s1[20];
 Student s;
 int i;
 printf("\nFisier: ");gets(s1);
 f=fopen(s1,"wb");
 printf("Nr.matricol: ");scanf("%d",&s.nr);
 while(!feof(stdin))
  {printf("Nume:        ");fflush(stdin);gets(s.nume);
   printf("An:          ");scanf("%d",&s.an);
   printf("Grupa:       ");scanf("%d",&s.grupa);
   printf("Nr.note:(<15)");scanf("%d",&s.n);
   for(i=0;i<s.n;i++)
    {printf("Nota %d:     ",i+1);
     scanf("%d",&s.note[i]);}
   fwrite(&s,sizeof(Student),1,f);
   printf("Nr.matricol: ");scanf("%d",&s.nr);}
 fclose(f);}
```

ii. Să se scrie programul care adaugă noi studenţi în fişierul creat la problema anterioară. Datele se preiau de la tastatură, sfîrşitul introducerii fiind marcat standard.

```
#include<stdio.h>
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f;
```

```
 char s1[20];
 Student s;
 int i;
 printf("\nFisier: ");gets(s1);
 f=fopen(s1,"rb+");
 fseek(f,0,SEEK_END);
 printf("Nr.matricol: ");scanf("%d",&s.nr);
 while(!feof(stdin))
  {printf("Nume:        ");fflush(stdin);gets(s.nume);
   printf("An:          ");scanf("%d",&s.an);
   printf("Grupa:       ");scanf("%d",&s.grupa);
   printf("Nr.note:(<15)");scanf("%d",&s.n);
   for(i=0;i<s.n;i++)
    {printf("Nota %d:     ",i+1);
     scanf("%d",&s.note[i]);}
   fwrite(&s,sizeof(Student),1,f);
   printf("Nr.matricol: ");scanf("%d",&s.nr);}
 fclose(f);
}
```

iii. Să se scrie programul care listează, într-un fişier text, sub formă de tabel, conținutul fişierului creat la problema de la punctul i.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f,*g;
 char s1[20];
 Student s;
 int i,n;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nFisier rezultat (text): ");gets(s1);
      g=fopen(s1,"w");
      fprintf(g,"\nNr. Nume %25s An Grupa Note"," ");
      freadb(s,f);n=0;
      while(!feof(f))
       {fprintf(g,"\n%3d %-30s %2d %4d  ",++n,s.nume,s.an,s.grupa);
        for(i=0;i<s.n;i++)
          fprintf(g,"%2d ",s.note[i]);
        freadb(s,f);
        }
      fclose(g);
      fclose(f);}
}
```

iv. Să se scrie programul care afişează datele despre studenţii ale căror numere matricole se introduc de la tastatură. Sfîrşitul introducerii este marcat standard.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;;
void main()
{FILE *f,*g;
 char s1[20];
 Student s;
 int i,n,j;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis", s1);
 else{printf("\nNr. matricol: ");
      scanf("%d",&n);
      while(!feof(stdin))
      {rewind(f);
       freadb(s,f);i=0;
       while((!feof(f))&&(!i))
        {if(n==s.nr)
          {i=1;
           printf("\nNr.mat:%3d Nume: %-30s An: %2d Grupa: %4d\nNote: ",
                    s.nr,s.nume,s.an,s.grupa);
           for(j=0;j<s.n;j++)
           printf("%2d ",s.note[j]);}
           freadb(s,f);}
         if(!i)printf("\nNu a fost gasit.");
         printf("\nNr. matricol: ");
         scanf("%d",&n);}
      fclose(f);}
}
```

v. Să se scrie programul care listează, în fişiere text, situaţia studenţilor din grupele ale căror numere se introduc de la tastatură. Sfîrşitul introducerii este marcat standard.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
```

```
void main()
{FILE *f,*g;
 char s1[20];
 Student s;
 int i,n,j;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nNr. grupei: ");
      scanf("%d",&n);
      while(!feof(stdin))
       {rewind(f);
        fflush(stdin);
        printf("\nFisier rezultat: ");gets(s1);
        g=fopen(s1,"w");
        freadb(s,f);i=0;
        while(!feof(f))
         {if(n==s.grupa)
           {i=1;
       fprintf(g,"\nNr.mat:%3d Nume: %-30s An: %2d Grupa: %4d\nNote: ",
                    s.nr,s.nume,s.an,s.grupa);
            for(j=0;j<s.n;j++)
              fprintf(g,"%2d ",s.note[j]);}
          freadb(s,f);}
        if(!i)printf("\nNu a fost gasita.");
        printf("\nNr. grupei: ");
        scanf("%d",&n);
        fclose(g);}
      fclose(f);}
}
```

vi. Să se scrie programul care sortează studenții după anii de studiu şi grupe.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
int nrart(FILE *f, int l)
{long p;
 int n;
 p=ftell(f);
 fseek(f,0,2);
 n=ftell(f)/l;
 fseek(f,0,p);
 return n;}
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f,*g;
 char s1[20];
 Student s,s2;
```

```
 int i,n,j;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{n=nrart(f,sizeof(Student));
      j=1;
      while(j)
       {j=0;
        for(i=0;i<n-1;i++)
         {fseek(f,i*sizeof(Student),SEEK_SET);
          freadb(s,f);
          freadb(s2,f);
          if((s.an>s2.an)||((s.an==s2.an)&&(s.grupa>s2.grupa)))
           {j=1;
            fseek(f,i*sizeof(Student),SEEK_SET);
            fwriteb(s2,f);
            fwriteb(s,f);}
         }
       }
      fclose(f);}
}
```

vii. Să se scrie programul care afişează studenţii integralişti din grupele ale căror numere sînt introduse de la tastatură. Sfîrşitul introducerii este marcat standard.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f,*g;
 char s1[20];
 Student s;
 int i,n,j,e;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nNr. grupei: ");
      scanf("%d",&n);
      while(!feof(stdin))
       {rewind(f);
        fflush(stdin);
        freadb(s,f);
        i=0;
        while(!feof(f))
         {if(n==s.grupa)
           {e=1;
            for(j=0;j<n;j++)
              if(s.note[j]<5)e=0;
```

```
           if(e)
             {i=1;
        printf("\nNr.mat:%3d Nume: %-30s An: %2d Grupa: %4d\nNote: ",
                  s.nr,s.nume,s.an,s.grupa);
              for(j=0;j<s.n;j++)
                printf("%2d ",s.note[j]);}
           }
          freadb(s,f);}
        if(!i)printf("\nNu au fost gasiti studenti integralisti/Nu
                        exista grupa.");
        printf("\nNr. grupei: ");
        scanf("%d",&n);
        fclose(g);}
      fclose(f);}
}
```

viii. Să se scrie programul care listează, într-un fişier text, studenții integralişti cu cea mai mare medie.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f,*g;
 char s1[20];
 Student s;
 int i,n,j,e;
 float m,max;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else {fflush(stdin);
       printf("\nFisier rezultat: ");gets(s1);
       g=fopen(s1,"w");
       freadb(s,f);
       i=0;  max=0;
       while(!feof(f))
        {m=0.0;e=1;
         for(j=0;j<s.n;j++)
         if(s.note[j]<5)e=0;
         else m+=s.note[j];
         if(e)
          {i=1;
           m=m/s.n;
           if(m>max){max=m;
                     g=fopen(s1,"w");
        fprintf(g,"\nNr.mat:%3d Nume: %-30s An: %2d Grupa: %4d Media:
                     %5.2f ",s.nr,s.nume,s.an,s.grupa,max);}
```

```
          else if(m==max)
         fprintf(g,"\nNr.mat:%3d Nume: %-30s An: %2d Grupa: %4d
                    Media: %5.2f",s.nr,s.nume,s.an,s.grupa,max);}
         freadb(s,f);}
       if(!i)printf("\nNu au fost gasiti studenti integralisti.");
       fclose(g);
       fclose(f);}
}
```

ix. Să se scrie programul care calculează nota medie la filosofie a studenților din anul 2, pe grupe. Se ştie că nota la filosofie este pe poziția a treia. Rezultatele vor fi scrise într-un fişier text.

Pentru rezolvarea problemei, studenții trebuie să fie sortați după ani şi grupe. Se va folosi întîi programul de la problema *vi*.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;

void main()
{FILE *f,*g;
 char s1[20];
 Student s;
 int i,n,j,e,n1,ca,cg,an=2;
 float m,mg;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else {freadb(s,f);e=1;
       while((!feof(f))&&(e))
        {ca=s.an;
         if(s.an!=an)freadb(s,f);
         else{m=0;n=0;
              while((s.an==ca)&&(!feof(f)))
               {mg=0;n1=0;
                cg=s.grupa;
                while((cg==s.grupa)&&(ca==s.an)&&(!feof(f)))
                 {mg+=s.note[2];
                  n1++;
                  freadb(s,f);}
                mg=mg/n1;
                printf("\nGrupa %d, media: %5.2f",cg,mg);
                m+=mg;n+=n1;}
              printf("\nMedia anului %d este: %5.2f",an,m/n);
              e=0;}
        }
       fclose(f);}
}
```

x. Să se scrie programul care listează, într-un fişier text, studenţii integralişti, pe ani şi grupe, calculînd media fiecărei grupe şi a fiecărui an.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f,*g;
 char s1[20];
 Student s;
 int i,na,j,e,ng,ca,cg;
 float ma,mg,ms;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else {printf("\nFisier text: ");gets(s1);
       g=fopen(s1,"w");
       freadb(s,f);
       while(!feof(f))
        {ca=s.an;
         fprintf(g,"\n\nAnul %d",ca);
         ma=0;na=0;
         while((s.an==ca)&&(!feof(f)))
          {mg=0;ng=0;
           cg=s.grupa;
           fprintf(g,"\n\tGrupa %d",cg);
           while((cg==s.grupa)&&(ca==s.an)&&(!feof(f)))
            {e=1;ms=0;
             for(j=0;j<s.n;j++)
               if(s.note[j]<5)e=0;
               else ms+=s.note[j];
             if(e){mg+=ms/s.n;
                   ng++;
                   fprintf(g,"\n\t\t%4d %-30s Media %5.2f Note: ",
                                    s.nr,s.nume,ms/s.n);
                   for(j=0;j<s.n;j++)fprintf(g,"%2d ",s.note[j]);}
             freadb(s,f);}
           if(ng){mg=mg/ng;
                  fprintf(g,"\n\tGrupa %d, media: %5.2f",cg,mg);
                  ma+=mg;na+=ng;}
           else fprintf(g,"\n\Grupa nu are integralisti");}
         if(na)fprintf(g,"\nMedia anului %d este: %5.2f",ca,ma/na);
         else fprintf(g,"\nAnul nu are integralisti");}
       fclose(f);}
}
```

xi. Să se scrie programul care listează, într-un fişier text, studenţii cu mai mult de 2 restanţe.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f,*g;
 char s1[20];
 Student s;
 int i,n,j,e;
 float m,max;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else {fflush(stdin);printf("\nFisier rezultat: ");gets(s1);
       g=fopen(s1,"w");
       freadb(s,f);
       i=0;e=0;
       while(!feof(f))
        {n=0;
         for(i=0;i<s.n;i++)
         n+=(s.note[i]<5);
         if(n>2)
          {e++;
           fprintf(g,"\n%4d %-30s an %2d grupa %2d Note:",s.nr,
                     s.nume, s.an,s.grupa);
           for(i=0;i<s.n;i++)
             fprintf(g,"%2d ",s.note[i]);}
         freadb(s,f);}
       fprintf(g,"\nTotal: %d studenti",e);
       fclose(g);
       fclose(f);}
}
```

xii. Să se scrie programul pentru modificarea notei la filosofie pentru studenţii din grupa al cărei număr este introdus de la tastatură.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
```

```
void main()
{FILE *f;
 char s1[20],materie[]="Filosofie";
 Student s;
 int i,n,j,e,nota=2;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb+")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nNr. grupei: ");
      scanf("%d",&n);
      while(!feof(stdin))
      {rewind(f);
       freadb(s,f);
       i=0;
       while(!feof(f))
       {if(n==s.grupa)
         {i=1;
          printf("\n %4d %-30s Nota la %s: %2d",s.nr,s.nume,materie,
                     s.note[nota]);
          printf("\n Noua nota(sau 0):");
          scanf("%d",&j);
          if(j){s.note[nota]=j;
                fseek(f,ftell(f)-sizeof(Student),SEEK_SET);
                fwriteb(s,f);fseek(f,0,1);}
         }
        freadb(s,f);}
       if(!i)  printf("\n Nu a fost gasit nici un student");
       printf("\nNr grupei: ");scanf("%d",&n);}
      fclose(f);}
}
```

xiii. Să se scrie programul pentru adăugarea punctului din oficiu la nota la filosofie pentru studenții din grupa al cărei număr este introdus de la tastatură.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f;
 char s1[20],materie[]="Filosofie";
 Student s;
 int i,n,j,e,nota=2;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb+")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nNr. grupei: ");
      scanf("%d",&n);
      while(!feof(stdin))
```

```
      {rewind(f);
       freadb(s,f);
       i=0;
       while(!feof(f))
       {if(n==s.grupa)
         {i=1;
          s.note[nota]=s.note[nota]+(s.note[nota]<10);
          fseek(f,ftell(f)-sizeof(Student),SEEK_SET);
          fwriteb(s,f); fseek(f,0,1);}
        freadb(s,f);}
       if(!i)  printf("\n Nu a fost gasit nici un student");
       printf("\nNr grupei: ");scanf("%d",&n);}
      fclose(f);}
}
```

xiv. Să se scrie programul care modifică o notă pentru studenții ale căror numere matricole se introduc de la tastatură. De la tastatură se va introduce numărul notei care se modifică (de exemplu, pentru filosofie se va introduce 3).

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f;
 char s1[20];
 Student s;
 int i,n,j,e,nota=2;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb+")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nNr. matricol: ");
      scanf("%d",&n);
      while(!feof(stdin))
      {rewind(f);
       freadb(s,f);
       i=0;
       while(!feof(f))
       {if(n==s.nr)
         {i=1;
          printf("\n %4d %-30s Nota la %s: %2d",s.nr,s.nume, materie,
                    s.note[nota]);
          printf("\n Noua nota(sau 0):");
          scanf("%d",&j);
          if(j){s.note[nota]=j;
                fseek(f,ftell(f)-sizeof(Student),SEEK_SET);
                fwriteb(s,f);}
         }
```

```
       freadb(s,f);}
       if(!i)  printf("\n Nu a fost gasit studentul");
       printf("\nNr matricol: ");scanf("%d",&n);}
      fclose(f);}
}
```

5.2.4 Fişiere organizate relativ

i. Să se scrie programul care creează un fişier organizat relativ cu date despre studenții unei facultăți. Datele care se reţin despre studenţi sînt: numele, anul, grupa, numărul de note, notele (maxim 15). Cheia relativă a fişierului este numărul matricol al studentului. Datele se preiau de la tastatură, sfîrşitul introducerii fiind marcat standard.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
#include<stdio.h>
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
int nrart(FILE *f, int l)
{long p;
 int n;
 p=ftell(f);
 fseek(f,0,2);
 n=ftell(f)/l;
 fseek(f,p,0);
 return n;}
void main()
{FILE *f;
 char s1[20];
 Student x;
 int n,i;
 printf("\nFisier: ");gets(s1);
 f=fopen(s1,"wb+");
 printf("\nNr matricol: ");
 scanf("%d",&n);
 while(!feof(stdin))
 {if(n>=nrart(f,sizeof(Student)))
   {x.is=0;
    fseek(f,0,SEEK_END);
    for(i=nrart(f,sizeof(Student));i<=n;i++)
      fwriteb(x,f);}
    fseek(f,n*sizeof(Student),SEEK_SET);
    freadb(x,f);
    if(x.is) printf("\nExista deja un student cu acest numar
                        matricol");
```

```
    else{fseek(f,n*sizeof(Student),SEEK_SET);
         x.nr=n;
         printf("Nume: "); fflush(stdin);gets(x.nume);
         printf("An  : "); scanf("%d",&x.an);
         printf("Grupa:"); scanf("%d",&x.grupa);
         printf("Nr. note:"); scanf("%d",&x.n);
         for(i=0;i<x.n;i++)
          {printf("Nota %d: ",i+1);
           scanf("%d",&x.note[i]);}
         x.is=1; fwriteb(x,f);}
         printf("\nNr matricol: "); scanf("%d",&n);}
    fclose(f);}
```

ii. Să se scrie programul care adaugă noi studenţi în fişierul creat la problema anterioară. Datele se preiau de la tastatură, sfîrşitul introducerii fiind marcat standard.

Programul este identic cu cel de la problema *i*, modificîndu-se doar modul de deschidere a fişierului: se încearcă deschiderea acestuia cu modul *rb+* şi, dacă deschiderea nu reuşeşte, atunci se creează un fişier nou (deschidere cu modul *wb+*).

iii. Să se scrie programul care listează, într-un fişier text, sub formă de tabel, studenţii din fişierul creat la problema de la punctul i.

```
#include<stdio.h>
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f,*g;
 char s1[20];
 Student x;
 int i,j;
 printf("\nFisier: "); gets(s1);
 if(!(f=fopen(s1,"rb+")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nFisier text: "); gets(s1);
      g=fopen(s1,"w"); i=0;
      fprintf(g,"\n Nrc Nrm Nume si prenume %15s An Grupa Note"," ");
      freadb(x,f);
      while(!feof(f))
       {if(x.is)
         {fprintf(g,"\n%4d %4d %-30s %2d %3d    ",++i,x.nr,x.nume,
                     x.an,x.grupa);
          for(j=0;j<x.n;j++)
          fprintf(g,"%2d ",x.note[j]);}
        freadb(x,f);}
      fclose(f);}
}
```

iv. Să se scrie programul care afișează datele despre studenții ale căror numere matricole se introduce de la tastatură. Sfîrşitul introducerii este marcat standard.

```
#include<stdio.h>
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
int nrart(FILE *f, int l)
{long p;
 int n;
 p=ftell(f);
 fseek(f,0,2);
 n=ftell(f)/l;
 fseek(f,p,0);
 return n;}
void main()
{FILE *f;
 char s1[20];
 Student x;
 int n,i;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nNr.matricol: ");
      scanf("%d",&n);
      while(!feof(stdin))
       {if(n>=nrart(f,sizeof(Student)))printf("\n Nu exista student
                                        cu numarul matricol %d",n);
        else{fseek(f,n*sizeof(Student),SEEK_SET);
             freadb(x,f);
             if(!x.is)printf("\n Nu exista student cu numarul matricol
                              %d",n);
             else{printf("\n %-30s An: %2d Grupa: %2d Note: ",x.nume,
                         x.an,x.grupa);
                  for(i=0;i<x.n;i++)
                    printf("%2d ",x.note[i]);}
            }
        printf("\nNr.matricol: ");
        scanf("%d",&n);}
      fclose(f);}
}
```

v. Să se scrie programul care listează, în fişiere text, situația studenților din grupele ale căror numere se introduc de la tastatură. Sfîrşitul introducerii este marcat standard.

```
#include<stdio.h>
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
```

```
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f,*g;
 char s1[20];
 Student x;
 int n,i,j;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nNr.grupei: ");
      scanf("%d",&n);
      while(!feof(stdin))
       {i=0;
        printf("\nFisier text: ");fflush(stdin);gets(s1);
        g=fopen(s1,"w");
        fprintf(g,"\n Nrc Nrm  Nume si prenume %14s An Grupa Note"," ");
        rewind(f);
       freadb(x,f);
       while(!feof(f))
        {if(x.is)
           if(x.grupa==n)
             {fprintf(g,"\n%4d %4d %-30s %2d %3d    ",++i,x.nr,
                           x.nume,x.an, x.grupa);
              for(j=0;j<x.n;j++)
                fprintf(g,"%2d ",x.note[j]);}
         freadb(x,f);}
       fclose(g);
       printf("\nAu fost listati %d studenti in fisierul %s\n\n\n",
                 i,s1);
       printf("\nNr.grupei: ");
       scanf("%d",&n);}
      fclose(f);}
}
```

vi. Să se scrie programul pentru exmatricularea studenților ale căror numere matricole se introduc de la tastatură.

```
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
int nrart(FILE *f, int l)
```

```
{long p;
 int n;
 p=ftell(f);
 fseek(f,0,2);
 n=ftell(f)/l;
 fseek(f,0,p);
 return n;}
void main()
{FILE *f;
 char s1[20];
 Student x;
 int n,i;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nNr.matricol: ");
      scanf("%d",&n);
      while(!feof(stdin))
       {if(n>=nrart(f,sizeof(Student)))printf("\n Nu exista student
                                          cu numarul matricol %d",n);
        else{fseek(f,n*sizeof(Student),SEEK_SET);
             freadb(x,f);
             if(!x.is)printf("\n Nu exista student cu numarul
                                matricol %d",n);
             else{printf("\n %-30s An: %2d Grupa: %2d Note: ",x.nume,
                           x.an,x.grupa);
                  for(i=0;i<x.n;i++)
                  printf("%2d ",x.note[i]);
                  printf("\nExmatriculez? (d/n): ");
                  if(toupper(getche())=='D')
                   {fseek(f,n*sizeof(Student),SEEK_SET);
                    x.is=0;
                    fwriteb(x,f);}
                 }
            }
      printf("\nNr.matricol: ");
      scanf("%d",&n);}
      fclose(f);}
}
```

vii. Să se scrie programul care afişează studenții integralişti din grupele ale căror numere sînt introduse de la tastatură. Sfîrşitul introducerii este marcat standard.

```
#include<stdio.h>
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f;
```

```
 char s1[20];
 Student x;
 int n,i,j,e;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nNr.grupei: ");
      scanf("%d",&n);
      while(!feof(stdin))
       {i=0;
        printf("\n Nrc Nrm  Nume si prenume %14s An Grupa Note"," ");
        rewind(f);
        freadb(x,f);
        while(!feof(f))
         {if(x.is)
            if(x.grupa==n)
              {e=1;
               for(j=0;j<x.n;j++)
                 if(x.note[j]<5)e=0;
               if(e){printf("\n%4d %4d %-30s %2d %3d    ",++i,x.nr,
                                 x.nume, x.an,x.grupa);
                     for(j=0;j<x.n;j++)
                      printf("%2d ",x.note[j]);}
              }
          freadb(x,f);}
        if(i)printf("\nAu fost listati %d studenti din grupa
                      %d\n\n\n",i,n);
        else printf("\nNici un student integralist/nu exista grupa");
        printf("\nNr.grupei: ");
        scanf("%d",&n);}
      fclose(f);}
}
```

viii. Să se scrie programul care listează, într-un fişier text, studenţii integralişti cu cea mai mare medie.

```
#include<stdio.h>
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f,*g;
 char s1[20];
 Student x;
 int j,e;
 float max,m;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nFisier text: ");
      gets(s1);
```

```
      g=fopen(s1,"w");
      max=0;
      fprintf(g,"\nNrm  Nume si prenume %14s An Grupa Media Note"," ");
      freadb(x,f);
      while(!feof(f))
       {if(x.is)
         {e=1;m=0.0;
          for(j=0;j<x.n;j++)
            if(x.note[j]<5)e=0;
            else m+=x.note[j];
          m=m/x.n;
        if(e)
          if(m>max)
           {fclose(g);
            g=fopen(s1,"w");
            max=m;}
            if(m==max)
             {fprintf(g,"\nNrm  Nume si prenume %14s An Grupa Media
                         Note"," ");
              fprintf(g,"\n%4d %-30s %2d %3d   %5.2f ",x.nr,x.nume,
                         x.an, x.grupa,max);
              for(j=0;j<x.n;j++)
                fprintf(g,"%2d ",x.note[j]);}
           }
        freadb(x,f);}
      fclose(f);}
}
```

ix. Să se scrie programul care calculează nota medie la filosofie a studenților din anul 2, pe grupe. Se ştie că nota la filosofie este pe poziția a treia. Rezultatele vor fi scrise într-un fişier text.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
int nrart(FILE *f, int l)
{long p;
 int n;
 p=ftell(f);
 fseek(f,0,2);
 n=ftell(f)/l;
 fseek(f,p,0);
 return n;}

void main()
{FILE *f,*g,*h;
 char s1[20],materie[]="Filosofie";
```

```
 Student x,y;
 int n,e,i,gr,an=2,poz=3;
 float m;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nFisier text: ");
      gets(s1);
      g=fopen(s1,"w");
      h=fopen("temp.dat","wb");
      //selectare studenti an 2
      freadb(x,f);
      while(!feof(f))
       {if(x.is&&(x.an==an)) fwriteb(x,h);
        freadb(x,f);}
      fclose(f);
      fclose(h);
      h=fopen("temp.dat","rb+");
      n=nrart(h,sizeof(Student));
      //sortare pe grupe
      e=1;
      while(e)
       {e=0;
        for(i=0;i<n-1;i++)
         {fseek(h,i*sizeof(Student),SEEK_SET);
          freadb(x,h);
          freadb(y,h);
          if(x.grupa>y.grupa)
           {fseek(h,i*sizeof(Student),SEEK_SET);
            fwriteb(y,h);
            fwriteb(x,h);
            e=1;}
         }
       }
       //listare
       rewind(h);
       fprintf(g,"\nAnul %d, %s",an,materie);
       freadb(x,h);
       while(!feof(h))
        {gr=x.grupa;
         fprintf(g,"\nGrupa %d",gr);
         m=0;i=0;
         while((!feof(h))&&(x.grupa==gr))
          {m+=x.note[poz-1];
           i++;
           freadb(x,h);}
         fprintf(g,"\tmedia %5.2f",m/i);}
       fclose(g);
       fclose(h);}
}
```

x. Să se scrie programul care listează într-un fişier text studenţii pe ani şi grupe, calculînd media fiecărui student, a fiecărei grupe şi a fiecărui an.

```
#include<stdio.h>
```

```
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
int nrart(FILE *f, int l)
{long p;
 int n;
 p=ftell(f);
 fseek(f,0,2);
 n=ftell(f)/l;
 fseek(f,0,p);
 return n;}
void main()
{FILE *f,*g,*h;
 char s1[20];
 Student x,y;
 int n,e,i,gr,an,na,ng,ns;
 float ma,mg,ms;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nFisier text: ");
      gets(s1);
      g=fopen(s1,"w");
      h=fopen("temp.dat","wb");
      //copiere studenti
      freadb(x,f);
      while(!feof(f))
       {if(x.is) fwriteb(x,h);
        freadb(x,f);}
      fclose(f);
      fclose(h);
      h=fopen("temp.dat","rb+");
      n=nrart(h,sizeof(Student));
      //sortare pe ani si grupe
      e=1;
      while(e)
       {e=0;
        for(i=0;i<n-1;i++)
         {fseek(h,i*sizeof(Student),SEEK_SET);
          freadb(x,h);
          freadb(y,h);
          if((x.an>y.an)||((x.an==y.an)&&(x.grupa>y.grupa)))
           {fseek(h,i*sizeof(Student),SEEK_SET);
            fwriteb(y,h);
            fwriteb(x,h);
            e=1;}
         }
       }
      //listare
      rewind(h);
```

```
      fprintf(g,"\nLista cu mediile studentilor\n\n");
      freadb(x,h);
      while(!feof(h))
       {an=x.an;
        fprintf(g,"\nAnul %d",an);
        ma=0;na=0;
        while((!feof(h))&&(an==x.an))
         {gr=x.grupa;
          fprintf(g,"\n\tGrupa %d",gr);
          mg=0;ng=0;
          while((!feof(h))&&(x.grupa==gr)&&(x.an==an))
           {ms=0;
            for(i=0;i<x.n;i++)
            ms+=x.note[i];
            ms=ms/x.n;
            mg+=ms;ng++;
            fprintf(g,"\n\t\t %4d %-30s Media: %5.2f",x.nr,x.nume,ms);
            freadb(x,h);}
          mg=mg/ng;
          fprintf(g,"\n\tMedia grupei %d este %5.2f",gr,mg);
          ma+=mg;na+=1;}
        ma=ma/na;
        fprintf(g,"\nMedia anului %d este %5.2f",an,ma);}
      fclose(g);
      fclose(h);
      unlink("temp.dat");}
}
```

xi. Să se scrie programul care listează, într-un fişier text, studenţii cu mai mult de 2 restanţe.

```
#include<stdio.h>
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f,*g;
 char s1[20];
 Student x;
 int n,i,e;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb+")))printf("\nFisierul    %s    nu    poate    fi
deschis",s1);
 else{printf("\nFisier text: ");gets(s1);
      g=fopen(s1,"w");
      n=0;
      fprintf(g,"\n Nrc  Nrm Nume si prenume %14s An Grupa Note"," ");
      freadb(x,f);
      while(!feof(f))
```

```
       {if(x.is)
         {e=0;
          for(i=0;i<x.n;i++)
            if(x.note[i]<5) e++;
          if(e>2)
           {fprintf(g,"\n%4d %4d %-30s %2d %3d    ",++n,x.nr,x.nume,
                        x.an,x.grupa);
            for(i=0;i<x.n;i++)
              fprintf(g,"%2d ",x.note[i]);}
         }
        freadb(x,f);}
      printf("\nAu fost listati %d studenti",n);
      fclose(f);}
}
```

xii. Să se scrie programul pentru exmatricularea studenților cu mai mult de trei restanțe.

```
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
void main()
{FILE *f;
 char s1[20];
 Student x;
 int n,i,e;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb+")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{n=0;
      freadb(x,f);
      while(!feof(f))
       {if(x.is)
         {e=0;
          for(i=0;i<x.n;i++)
          if(x.note[i]<5) e++;
            if(e>3)
             {printf("\n\n%4d %-30s %2d %3d    ",x.nr,x.nume,x.an,
                         x.grupa);
              for(i=0;i<x.n;i++)
                printf("%2d ",x.note[i]);
              printf("\nExmatriculez? (d/n): ");
              if(toupper(getche())=='D')
               {fseek(f,ftell(f)-sizeof(Student),SEEK_SET);
                x.is=0;
                n++;
```

```
                fwriteb(x,f);}
             }
          }
        freadb(x,f);}
      fclose(f);
      printf("\nAu fost exmatriculati %d studenti",n);}
}
```

xiii. Să se scrie programul pentru modificarea notei la filosofie pentru studenții din grupa al cărei număr este introdus de la tastatură.

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa; int n;
               int note[15];
              }Student;
void main()
{FILE *f;
 char s1[20],materie[]="Filosofie";
 Student x;
 int n=0,i,j,poz=3;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb+")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nNr.grupei: ");
      scanf("%d",&n);
      while(!feof(stdin))
       {rewind(f);
        j=0;
        freadb(x,f);
        while(!feof(f))
         {if((x.is)&&(x.grupa==n))
           {j++;
            printf("\n\n%4d %-30s %2d %2d Nota la %s este: %2d",x.nr,
                          x.nume,x.an,x.grupa,materie,x.note[poz-1]);
            printf("\nNoua nota: ");
            scanf("%d",&i);
            if(i){fseek(f,ftell(f)-sizeof(Student),SEEK_SET);
                  x.note[poz-1]=i;
                  fwriteb(x,f);
                  fseek(f,0,SEEK_CUR);}
           }
          freadb(x,f);}
        printf("\n Au fost gasiti %d studenti",j);
        printf("\nNr.grupei: ");
        scanf("%d",&n);}
      fclose(f);}
}
```

xiv. Să se scrie programul care modifică o notă pentru studenții ale căror numere matricole se introduc de la tastatură. De la tastatură se va introduce numărul notei care se modifică (de exemplu, pentru filosofie se va introduce 3).

```
#include<stdio.h>
#define fwriteb(x,f) fwrite(&(x),sizeof(Student),1,(f))
#define freadb(x,f) fread(&(x),sizeof(Student),1,(f))
typedef struct{int nr;
               char nume[30];
               int an;
               int grupa;
               int n;
               int note[15];
              }Student;
int nrart(FILE *f, int l)
{long p; int n;
 p=ftell(f);
 fseek(f,0,2); n=ftell(f)/l;
 fseek(f,p,0);
 return n;}
void main()
{FILE *f;
 char s1[20]; Student x;
 int n,i,j,e;
 printf("\nFisier: ");gets(s1);
 if(!(f=fopen(s1,"rb+")))printf("\nFisierul %s nu poate fi deschis",s1);
 else{printf("\nNr.matricol: ");
      scanf("%d",&n);
      while(!feof(stdin))
       {if(n>=nrart(f,sizeof(Student)))printf("\nNu exista studentul
                                          cu numarul matricol %d",n);
        else{fseek(f,n*sizeof(Student),SEEK_SET);
             freadb(x,f);
             if(!x.is)printf("\nNu exista studentul cu numarul
                              matricol %d",n);
             else{printf("\n\n%4d %-30s %2d %2d Note: ",x.nr,x.nume,
                           x.an,x.grupa);
                  for(i=0;i<x.n;i++)
                    printf("%2d ",x.note[i]);
                  do{printf("\nSe modifica nota cu numarul: ");
                     scanf("%d",&j);}
                  while((j<0)||(j>x.n));
                  do{printf("\nNoua nota: ");
                     scanf("%d",&e);}
                  while((e<0)||(e>x.n));
                  x.note[j-1]=e;
                  fseek(f,n*sizeof(Student),SEEK_SET);
             fwriteb(x,f);}
       }
       printf("\nNr.matricol: ");
       scanf("%d",&n);}
      fclose(f);}}
```

B I B L I O G R A F I E

1. [Negr94]
   Negrescu, L. *Limbajele C şi C++ pentru începători*, vol. I, *Limbajul C*, Editura Microinformatica, Cluj-Napoca, 1994

2. [Smeu95]
   Smeureanu, I., Ivan, I., Dârdală, M. *Limbajul C/C++ prin exemple*, Editura Cison, Bucureşti, 1995

3. [Cris97]
   Cristea, V., Athanasiu, I., Kalisz, E., Iorga, V. *Tehnici de programare*, Editura Teora, Bucureşti, 1997

4. [Kern78]
   Kernighan, B., Richie, D. *The C Programming Language*, Prentice-Hall, New Jersey, 1978

5. [Roşc94]
   Roşca, I., Gh., Apostol, C., Ghilic-Micu, B., Roşca, V. *Prelucrarea fişierelor în Pascal*, Editura Tehnică, Bucureşti, 1994

6. [Roşc96]
   Roşca, I., Gh., Apostol, C., Ghilic-Micu, B., Roşca, V. *Introducere în programare. Teorie şi practică Pascal*, Casa de editură şi presă Viața românească, Bucureşti, 1996

7. [Roşc98]
   Roşca, Ion Gh., Apostol, C., Ghilic-Micu, B., Roşca, V. *Programare sistematică în Pascal*, Editura Didactică şi Pedagogică, Bucureşti, 1998

8. [Roşc98]
   Roşca, I., Gh., Apostol, C., Ghilic-Micu, B., Stoica M., M., Cocianu, C., Uscatu, C. *Bazele elaborării programelor*, Editura A.S.E., Bucureşti, 1999

9. [Stro91]
   Stroustrup, B. *The C++ Programming Language*, AT & T, S.U.A., 1991